Theoretical Computer Science 60 (1988) 341-381
North-Holland



# LA RECONNAISSANCE DES FACTEURS D'UN LANGAGE FINI DANS UN TEXTE EN TEMPS LINEAIRE

Jean-Claude SPEHNER

*Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université de Haute Alsace, 68093 Mulhouse Cedex, France*

Communicated by M. Nivat
Received November 1987

**Résumé.** Nous donnons ici premièrement une construction en ligne d'un transducteur $\mathfrak{F}(L)$ qui reconnaît tous les facteurs d'un langage fini $L$ et place chacun d'eux comme facteur d'un des mots de $L$. $\mathfrak{F}(L)$ peut être deux fois plus petit que l'automate partiel introduit par A. Blumer, J. Blumer, D. Haussler, R. McConnell et A. Ehrenfeucht [6, 8] et qui reconnaît les mêmes mots. Par contre la construction de $\mathfrak{F}(L)$ est en $O(\|L\| \cdot (|A| + \min(|L|, \text{lgmax})))$ où $|L|$ et $|A|$ sont les cardinaux respectifs de $L$ et de son alphabet $A$, $\|L\|$ est la somme des longueurs des mots de $L$ et lgmax est la longueur maximum des mots de $L$ et non en $O(\|L\|)$ comme le leur.

Nous construisons ensuite un second transducteur $\mathfrak{F}'(L)$ qui admet les mêmes états que $\mathfrak{F}(L)$ et qui, pour tout facteur $u$ de $L$ et pour toute lettre $a$ de $A$ tels que $ua$ ne soit pas facteur de $L$, détermine et place le plus grand facteur droit de $ua$ qui soit facteur de $L$. Ce transducteur généralise celui que nous avons introduit en [20] pour un mot unique et sa construction est en $O(\|L\| \cdot |A|)$.

A l'aide des transducteurs $\mathfrak{F}(L)$ et $\mathfrak{F}'(L)$ nous obtenons un algorithme qui détermine toutes les occurrences de facteurs de L dans un texte linéairement par rapport à la longueur du texte et indépendamment de l'alphabet du texte.

Cet algorithme peut être utilisé par un informaticien pour retrouver et modifier toute une famille d'identificateurs dans un programme. Un linguiste peut aussi déterminer tous les mots d'une même famille ou relatifs à un même concept en éliminant éventuellement les mots paronymes.

**Abstract.** First we give here an on-line construction of a transducer $\mathfrak{F}(L)$ which recognizes all the factors of a finite language $L$ and positions each factor as a factor of a word from $L$. $\mathfrak{F}(L)$ can be twice smaller than the partial automaton introduced by A. Blumer, J. Blumer, D. Haussler, R. McConnel and A. Ehrenfeucht [6, 8] which recognizes the same words. Though, the complexity of the construction of $\mathfrak{F}(L)$ is in $O(\|L\| \cdot (|A| + \min(|L|, \text{lgmax})))$ where $|L|$ and $|A|$ are respectively the cardinality of $L$ and of its alphabet $A$, $\|L\|$ is the sum of the lengths of the words from $L$ and lgmax is the maximal length of these words and not in $O(\|L\|)$.

Then we build a second transducer $\mathfrak{F}'(L)$ which has the same states as $\mathfrak{F}(L)$ and which finds, for each factor $u$ of $L$ and each letter $a$ of $A$ such that $ua$ is not a factor of $L$, the largest right factor of $ua$ which is a factor of $L$. $\mathfrak{F}'(L)$ generalizes the transducer we have introduced in [20] for a unique word. The determination of $\mathfrak{F}'(L)$ is in $O(\|L\| \cdot |A|)$.

By using the transducers $\mathfrak{F}(L)$ and $\mathfrak{F}'(L)$, we obtain an algorithm which finds all the occurrences of the factors of $L$ in a text in time linear in the length of the text and independently of the cardinality of the alphabet of this text.

This algorithm can be used in computing to find and modify a family of identifiers in a program. Linguists can also determine all the words of a same family or related to a same concept—paronym words may be eliminated.

## Table des matières



## Introduction

Le problème de la recherche d'un mot dans un texte appelé "string matching" en anglais est résolu en temps linéaire relativement à la longueur du texte par les algorithmes de Knuth, Morris et Pratt [16] (voir aussi [18]) ou de Boyer et Moore [9] (voir aussi [1, 2, 17, 22]).

A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler et R. McConnell ont donné en [4] un algorithme qui détermine l'automate partiel minimal des facteurs droits d'un mot $x$ puis en [5, 6] un algorithme qui détermine l'automate partiel minimal $\mathfrak{F}(x)$ des facteurs de $x$ en temps linéaire par rapport à la longueur $|x|$ du mot $x$ (voir aussi [7, 11]).

M. Crochemore [12] (voir aussi [13, 14]) a donné indépendamment un algorithme qui détermine aussi $\mathfrak{F}(x)$ et en outre une fonction de sortie qui détermine la position de la première occurrence de chaque facteur de $x$ dans le mot $x$: $\mathfrak{F}(x)$ est alors un transducteur (voir [3, 15]).

En [20] nous avons construit un second transducteur $\mathfrak{F}'(x)$ qui, pour chaque facteur $u$ de $x$ et pour chaque lettre $a$ de l'alphabet $A$ de $x$ tels que $ua$ ne soit pas facteur de $L$, détermine le plus grand facteur droit $v$ de $ua$ qui est aussi un facteur de $x$ et la position de la première occurrence de $v$ dans $x$. Nous obtenons ainsi un algorithme qui détermine les occurrences des facteurs de $x$ dans un texte en temps linéaire par rapport à la longueur du texte.

Les algorithmes précédents peuvent être utilisés pour le mot $x_1\$x_2\$\ldots\$x_t$ associé au langage $L=\{x_1, x_2, \ldots, x_t\}$ mais le transducteur ainsi obtenu est presque $t=|L|$ fois plus grand que l'automate partiel minimal de $L\$=\{x_1\$_1, x_2\$_2, \ldots, x_t\$_t\}$ (avec $\$_1, \ldots, \$_t$ deux à deux distincts) utilisé par A. Blumer, J. Blumer, D. Haussler, R. McConnell et A. Ehrenfeucht en [6] (voir aussi [8]).

Cet article présente une généralisation aux langages finis de nos résultats pour un mot unique [20]. En outre l'automate construit est toujours plus petit que celui de [6] et peut être utilisé pour répondre à toutes les questions qu'ils résolvent.

## 1. Généralités sur les automates qui reconnaissent les facteurs de $L$

**1.1. Définition.** Soient $A^*$ le monoïde libre d'alphabet $A$ et 1 son élément neutre. $\forall x \in A^*$, soit $|x|$ la longueur du mot $x$.

(i) $\forall x \in A^*$, un mot $u$ de $A^*$ est appelé un *facteur* (respectivement un *facteur gauche*, un *facteur droit*) *de* $x$ s'il existe $v_1, v_2 \in A^*$ tels que $x = v_1 u v_2$ (respectivement $x = uv_2$, $x = v_1 u$). Soit $F(x)$ (respectivement $\mathrm{Fg}(x)$, $\mathrm{Fd}(x)$) l'ensemble des facteurs (respectivement facteurs gauches, facteurs droits) de $x$. Si $L \subseteq A^*$, tout mot $u$ de $F(L) = \bigcup_{x \in L} F(x)$) (respectivement $\mathrm{Fg}(L) = \bigcup_{x \in L} \mathrm{Fg}(x)$, $\mathrm{Fd}(L) = \bigcup_{x \in L} \mathrm{Fd}(x)$) est appelé un *facteur* (respectivement *facteur gauche, facteur droit*) *de* $L$. Un *mot* $x$ de $L$ est dit *redondant* si $F(L \backslash \{x\}) = F(L)$.

(ii) $\forall u \in \mathrm{Fg}(x)$ (respectivement $\forall v \in \mathrm{Fd}(x)$), il existe un unique mot $v$ (respectivement $u$) de $A^*$ tel que $uv = x$ et ce mot est noté $v = u^{-1}x$ (respectivement $u = xv^{-1}$). Si $K \subseteq A^*$ et $u \in A^*$, $u^{-1}K = \{v \in A^*; uv \in K\}$ est l'ensemble des *contextes droits de* $u$ *pour* $K$. $\mathrm{Pd}(K) = \{(u, v) \in A^* \times A^*; u^{-1}K = v^{-1}K\}$ est la *congruence syntaxique à droite du langage* $K$; c'est la plus grande congruence à droite de $A^*$ qui *sature* $L$ ($\forall x \in L$, la classe de $x$ modulo $\mathrm{Pd}(L)$ est contenue dans $L$).

**1.2. Définition.** (i) On dit qu'un *automate* $\mathfrak{A} = (S, \tau)$ *reconnaît une famille* $\tilde{K} = (K_i)_{i \in I}$ *de langages* de $A^*$, s'il existe un état $s_0 \in S$ et une famille $(T_i)_{i \in I}$ de parties de $S$ tels que, $\forall i \in I$, $w \in L_i \Leftrightarrow \tau(s_0, w) \in T_i$.

(ii) $\mathrm{Pd}(\tilde{K}) = \bigcap_{i \in I} \mathrm{Pd}(K_i)$ est la plus grande congruence à droite de $A^*$ qui sature simultanément tous les langages $K_i$ ($i \in I$) et l'automate $\mathfrak{M}(\tilde{K}) = (A^*/\mathrm{Pd}(\tilde{K}), \tau)$ où $\tau$ vérifie, $\forall u, v \in A^*$, $\tau(\bar{u}, v) = \overline{uv}$, $\bar{u}$ et $\overline{uv}$ étant les classes respectives de $u$ et de $uv$ modulo $\mathrm{Pd}(\tilde{K})$ est appelé l'*automate minimal de* $\tilde{K} = (K_i)_{i \in I}$. $\mathfrak{M}(\tilde{K})$ reconnaît la famille de langages $\tilde{K}$ et si, $\forall i \in I$, $\mathfrak{A}_i$ est l'automate minimal du langage $K_i$ et $s_{0,i}$ son état initial, alors $\mathfrak{M}(\tilde{K})$ est isomorphe au sous-automate de l'automate produit de $(\mathfrak{A}_i)_{i \in I}$ engendré par l'état initial $(s_{0,i})_{i \in I}$. Voir [21].

(iii) Soit $L = (x_1, \ldots, x_t)$ une suite finie de mots de $A^*$ dont aucun mot n'est redondant et telle que $|x_1| \geqslant |x_2| \geqslant \cdots \geqslant |x_t|$. $\forall i \in \{1, \ldots, t\}$, soit $\check{x}_i$ le facteur gauche de $x_i$ obtenu en supprimant la dernière lettre de $x_i$ et soit $L_i = \mathrm{Fd}(x_i) \backslash F(x_1, \ldots, x_{i-1}, \check{x}_i, \ldots, \check{x}_t)$ et soit, $\forall u \in A^*$, $[u]_i$ la classe de $u$ modulo $\mathrm{Pd}(L_i)$ et $[u] = \bigcap_{i \in 1, \ldots, t} [u]_i$ la classe de $u$ modulo $\mathrm{Pd}(\tilde{L})$ avec $\tilde{L} = (L_1, L_2, \ldots, L_t)$. Soit $\mathfrak{M}(\tilde{L})$ l'automate minimal de $\tilde{L}$. Comme $\tilde{L}$ dépend de l'ordre des mots de $L$, il en est de même de l'automate $\mathfrak{M}(\tilde{L})$ comme le montre l'exemple qui suit.

**Exemple.** Soient les mots $x = abbabc$ et $y = bababc$.

(i) Si $L = (x, y)$, $L_1 = \mathrm{Fd}(x)$, $L_2 = \{bababc, ababc\}$ et l'automate partiel $\mathfrak{F}(L)$ obtenu en supprimant dans $\mathfrak{M}(\tilde{L})$ l'état qui reconnaît $A^* \backslash F(L)$ est donné par la Fig. 1.

(ii) Si $L' = (y, x)$, $L_1 = \mathrm{Fd}(y)$, $L_2 = \{abbabc, bbabc\}$ et l'automate partiel $\mathfrak{F}(L')$ obtenu en supprimant dans $\mathfrak{M}(\tilde{L}')$ l'état qui reconnaît $A^* \backslash F(L')$ est donné par la Fig. 2.

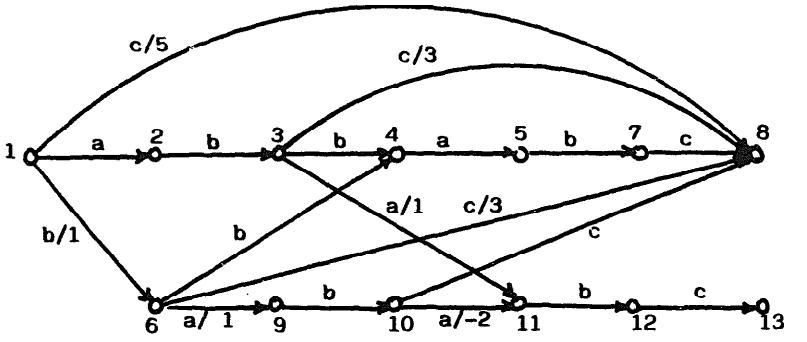


Fig. 1.

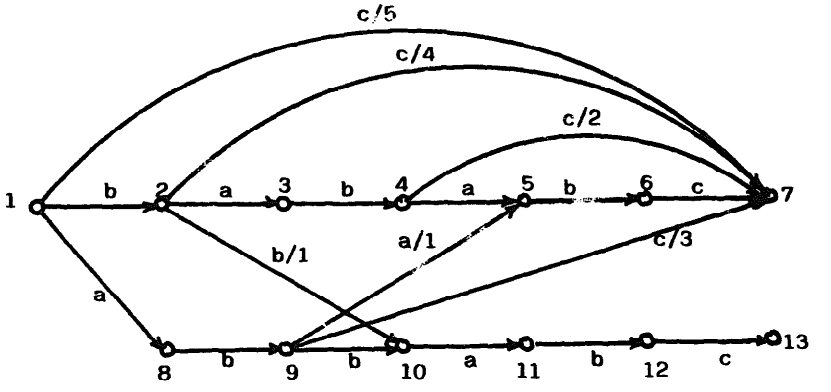


Fig. 2.

**1.3. Proposition.** (i) *L'automate* $\mathfrak{M}(\tilde{L})$ *reconnaît le langage* $F(L)$.

(ii) *L'automate minimal* $\mathfrak{M}(\tilde{K})$ *de* $\tilde{K}=(K_1,\dots,K_t)$ *où,* $\forall i\in\{1,\dots,t\}$, $K_i=F(x_i)\backslash F(x_1,\dots,x_{i-1})$ *est un automate quotient de l'automate minimal* $\mathfrak{M}(\tilde{J})$ *de* $\tilde{J}=(F(x_1),\dots,F(x_t))$ *et* $\mathfrak{M}(\tilde{L})$ *est un automate quotient de* $\mathfrak{M}(\tilde{K})$.

(iii) *Si* $\$_1,\$_2,\dots,\$_t$ *sont des lettres deux à deux distinctes et qui n'appartiennent pas à* $A$ *et si* $L_\$=\{x_1\$_1,x_2\$_2,\dots,x_t\$_t\}$, *l'automate minimal* $\mathfrak{M}(\tilde{D})$ *de* $\tilde{D}=(\mathrm{Fd}(x_1),\dots,\mathrm{Fd}(x_t))$ *est un automate quotient de l'automate minimal* $\mathfrak{M}(L_\$)$ *du langage* $L_\$$ *et comporte un unique état en moins et* $\mathfrak{M}(\tilde{J})$ *est un automate quotient de* $\mathfrak{M}(\tilde{D})$.

**Preuve.** (i): $\forall u\in F(L)$, il existe un mot $v$ de longueur maximum tel que $uv\in\mathrm{Fd}(L)$. Alors, si $L_0=\{\check{x}_1,\dots,\check{x}_{t-1},\check{x}_t\}$, $uv\notin F(L_0)$ et il existe un indice unique $i\in\{1,\dots,t\}$ tel que $uv\in\mathrm{Fd}(x_i)\backslash F(x_1,\dots,x_{i-1},\check{x}_i,\dots,\check{x}_t)=L_i$ et, $\forall u'\in[u]\subseteq[u]_i$, $u'v\in L_i$ et, par suite, $u'\in F(L)$ ce qui prouve que $[u]\subseteq F(L)$ et que $\mathfrak{M}(\tilde{L})$ reconnaît $F(L)$.

(ii): (1) $\forall i\in\{1,\dots,t\}$, $K_i\backslash L_i\subseteq F(L_0)$, $L_i=K_i\backslash F(\check{x}_i,\dots,\check{x}_t)$ et, $\forall a\in A$, $L_ia\subseteq A^*\backslash F(L)$ qui est une classe modulo $\mathrm{Pd}(\tilde{K})$. $L_i$ est donc aussi une classe modulo $\mathrm{Pd}(\tilde{K})$. Pd $(\tilde{K})$ sature donc les langages $L_1,\dots,L_t$ et, par suite, $\mathrm{Pd}(\tilde{K})\subseteq\mathrm{Pd}(\tilde{L})$ et $\mathfrak{M}(\tilde{L})$ est un automate quotient de $\mathfrak{M}(\tilde{K})$.

$\mathfrak{M}(\tilde{L})$ peut être strictement plus petit que $\mathfrak{M}(\tilde{K})$ comme on peut le voir sur l'exemple qui suit. Si $L = (abbcabc, abbabc, ababcb)$, $\mathfrak{M}(\tilde{K})$ est le complété (automate obtenu par l'adjonction d'un état puits) de l'automate partiel de la Fig. 4 (en la Section 3) et comporte 22 états alors que l'automate $\mathfrak{M}(\tilde{L})$ est le complété de celui de la Fig. 5 et comporte 18 états.

(ii): (2) Si $\tilde{J} = (F(x_1), \ldots, F(x_t))$, $\mathrm{Pd}(\tilde{J})$ sature les langages $F(x_1), \ldots, F(x_t)$ et, par suite, aussi $K_i = F(x_i) \backslash F(x_1, \ldots, x_{i-1})$, $\forall i \in \{1, \ldots, t\}$. Il en résulte que $\mathrm{Pd}(\tilde{J}) \subseteq \mathrm{Pd}(\tilde{K})$ et que $\mathfrak{M}(\tilde{K})$ est un automate quotient de $\mathfrak{M}(\tilde{J})$.

$\mathfrak{M}(\tilde{K})$ peut être strictement plus petit que $\mathfrak{M}(\tilde{J})$ comme on peut le voir sur l'exemple suivant. Si $x = abbabc$ et $y = bababc$ et si $L = (x, y)$ et $L' = (y, x)$, $\tilde{K} = \tilde{L} \neq \tilde{K}' = \tilde{L}'$, alors que $\tilde{J} = \tilde{J}'$ (à l'ordre près) et $\mathfrak{M}(\tilde{J})$ a un état de plus que $\mathfrak{M}(\tilde{K})$ et $\mathfrak{M}(\tilde{K}')$ provenant de l'intersection de la classe $L_1 = \mathrm{Fd}(x)$ modulo $\mathrm{Pd}(\tilde{L})$ avec la classe $L_1' = \mathrm{Fd}(y)$ modulo $\mathrm{Pd}(\tilde{L}')$.

(iii): (1) $\forall u \in F(L)$, soient $[u]_\$$, $[u]_D$ et, $\forall i \in \{1, \ldots, t\}$, $[u]_i'$ les classes respectives de $u$ modulo $\mathrm{Pd}(L_\$)$, $\mathrm{Pd}(\tilde{D})$ et $\mathrm{Pd}(\mathrm{Fd}(x_i))$. $\forall u \in [u]_\$$, $\forall i \in \{1, \ldots, t\}$ tel que $u \in F(x_i)$, $\forall w \in A^*$ tel que $uw \in \mathrm{Fd}(x_i)$, $uw\$_i \in L_\$$, $w\$_i \in u^{-1}L_\$ = v^{-1}L_\$$ d'où $vw\$_i \in L_\$$ et $vw \in \mathrm{Fd}(x_i)$ ce qui implique que $v \in F(x_i)$ et que $u^{-1}\mathrm{Fd}(x_i) \subseteq v^{-1}\mathrm{Fd}(x_i)$ d'où $u^{-1}\mathrm{Fd}(x_i) = v^{-1}\mathrm{Fd}(x_i)$ par raison de symétrie. En outre, si $u \notin F(x_i)$, alors $v \notin F(x_i)$ et $u^{-1}\mathrm{Fd}(x_i) = v^{-1}\mathrm{Fd}(x_i) = \emptyset$. Il en résulte que $[u]_\$ \subseteq [u]_i'$, $\forall i \in \{1, \ldots, t\}$ et, par suite, que $[u]_\$ \subseteq \bigcap_{i=1} [u]_i' = [u]_{\tilde{D}}$. L'automate minimal $\mathfrak{M}(L_\$)$ du langage $L_\$$ admet donc $\mathfrak{M}(\tilde{D})$ comme automate quotient.

Soit $\mathfrak{M}'(\tilde{D})$ l'automate obtenu à partir de $\mathfrak{M}(\tilde{D})$ en ajoutant un état supplémentaire $z$ et une $\forall i \in \{1, \ldots, t\}$, la transition $\tau'([x_i]_{\tilde{D}}, \$_i) = z$ et, $\forall a \in A$, la transition $\tau'(z, a) = z_0$ où $z_0$ est l'état qui reconnaît les mots de $A^* \backslash F(L)$. Alors $\mathfrak{M}(\tilde{D})$ reconnaît le langage $L_\$$ et est, d'après ce qui précède, isomorphe à l'automate minimal $\mathfrak{M}(L_\$)$ du langage $L_\$$. Cet automate $\mathfrak{M}(L_\$)$ est utilisé par [6].

(iii): (2) $\forall i \in \{1, \ldots, t\}$, l'automate minimal de $\mathrm{Fd}(x_i)$ reconnaît aussi le langage $F(x_i)$ en prenant comme états terminaux tous les états sauf celui qui reconnaît les mots de $A^* \backslash F(L)$. L'automate minimal $\mathfrak{M}(\tilde{D})$ reconnaît donc la famille de langages $\tilde{J} = (F(x_1), \ldots, F(x_t))$ et admet donc $\mathfrak{M}(\tilde{J})$ comme automate quotient.

D'après Crochemore [12], si $L$ est réduit à un unique mot $x = ab^n$ avec $a \neq b$, alors $\mathfrak{M}(\mathrm{Fd}(x))$ comporte $2n+2$ états, alors que $\mathfrak{M}(F(x))$ n'en comporte que $n+2$. □

**1.4. Proposition.** *Si $u \in f(L) \backslash \{1\}$ est le plus long mot de sa classe $[u]$ modulo $\mathrm{Pd}(\tilde{L})$ et si $\mathrm{suf}(u)$ est le plus long mot de $\mathrm{Fd}(u) \backslash [u]$, alors $[u] = \mathrm{Fd}(u) \backslash \mathrm{Fd}(\mathrm{suf}(u))$.*

**Preuve.** (i) Soit $\rho_0$ la relation d'équivalence sur $A^*$ dont les classes sont $L_1, \ldots, L_t$ et $A^* \backslash \bigcup_{i \in 1, \ldots, t} L_i$ et, $\forall i \in \mathbb{N}$, soit $\rho_{i+1}$ l'ensemble des couples $(u, v) \in \rho_i$ tels que, $\forall a \in A$, $(ua, va) \in \rho_i$. $\rho = \bigcap_{i \in \mathbb{N}} \rho_i$ est alors la plus petite congruence à droite de $A^*$ telle que $\rho \subseteq \rho_0$, c'est-à-dire qui sature $L_1, \ldots, L_t$ et, par suite, $\rho = \mathrm{Pd}(\tilde{L})$. Comme $\rho$ est d'index fini, il existe un entier naturel $r$ tel que

$$\rho_0 \supset \rho_1 \supset \cdots \supset \rho_{r-1} \supset \rho_r = \rho_{r+1} = \cdots \quad \text{et} \quad \mathrm{Pd}(\tilde{L}) = \rho_r.$$

(ii) $\forall K \subseteq F(L)$, soit $\ell(K) = \max\{|y|; (\exists k \in K)(\exists i \in \{1, \ldots, t\}) ky \in L_i\}$. Montrons par récurrence sur $i$ que toute classe $[u]$ modulo $\rho_i$ telle que $\ell([u]) \leqslant i$ est aussi une classe modulo $\mathrm{Pd}(\tilde{L})$ et est de la forme $\mathrm{Fd}(u) \backslash \mathrm{Fd}(\mathrm{suf}(u))$, lorsque $u$ est le plus long mot de $[u]$. Si, $\forall j \in \{1, \ldots, t\}$, $\mathrm{suf}(x_j)$ est le mot le plus long de $\mathrm{Fd}(x_j) \cap F(x_1, \ldots, x_{j-1}, \check{x}_j, \ldots, \check{x}_t)$, $L_j = \mathrm{Fd}(x_j) \backslash \mathrm{Fd}(\mathrm{suf}(x_j))$. En outre $\ell(L_j) = 0$ et, $\forall a \in A$, $L_j a \subseteq A^* \backslash F(L)$ qui est une classe modulo $\mathrm{Pd}(\tilde{L})$ ce qui prouve que $L_j$ est aussi une classe modulo $\mathrm{Pd}(\tilde{L})$ et que la propriété est vérifiée pour $i = 0$.

Supposons la propriété vérifiée pour $i$ et soit $[u]$ une classe modulo $\rho_{i+1}$ telle que $\ell([u]) \leqslant i+1$. Alors, $\forall a \in A$ tel que $ua \in F(L)$, $[u]a \subseteq [ua]$ où $[ua]$ est une classe modulo $\rho_i$ vérifiant $\ell([ua]) \leqslant i$ et, par suite, aussi une classe modulo $\mathrm{Pd}(\tilde{L})$ d'après l'hypothèse de récurrence. En outre, $\forall a \in A$ tel que $ua \notin F(L)$, $\forall v \in [u]$, $va \notin F(L)$ et, par suite, $[u]a \subseteq A^* \backslash F(L)$. Il en résulte que $[u]$ est aussi une classe modulo $\mathrm{Pd}(\tilde{L})$. Comme $(u, \mathrm{suf}(u)) \notin \rho_{i+1}$, il existe un plus grand indice $j \in \{0, \ldots, i\}$ tel que $(u, \mathrm{suf}(u)) \in \rho_j$ et il existe $a \in A$ tel que $(ua, \mathrm{suf}(u)a) \notin \rho_j$. Si $ua \in F(L)$, alors la classe $[ua]$ modulo $\rho_i$ est aussi une classe modulo $\mathrm{Pd}(\tilde{L})$ et, $\forall v \in \mathrm{Fd}(\mathrm{suf}(u))$, $va \notin [ua]$ d'après l'hypothèse de récurrence. Si $ua \notin F(L)$, $\mathrm{suf}(u)a \in F(L)$ et, $\forall v \in \mathrm{Fd}(\mathrm{suf}(u))$, $va \in F(L)$ d'où $v \notin [u]$. Dans les deux cas on trouve donc $[u] = \mathrm{Fd}(u) \backslash \mathrm{Fd}(\mathrm{suf}(u))$, lorsque $u$ est le plus long mot de $[u]$ et la proposition s'en déduit par récurrence. □

**1.5. Définition.** (i) $\forall i \in \{1, \ldots, t\}$ et $\forall u \in F(x_i)$, soit $\mathrm{pref}_i(u)$ le plus petit facteur gauche de $x_i$ qui admet $u$ comme facteur droit. $\forall u \in F(L)$, soit $i$ le plus petit entier de $\{1, \ldots, t\}$ tel que $u \in F(x_i)$ et soit $\mathrm{pref}(u) = \mathrm{pref}_i(u)$.

Si $u$ est le plus grand mot de $[u]$, $\forall u' \in [u]$, $u' \in \mathrm{Fd}(u)$ d'après la Proposition 1.4 et, comme $[u] \subseteq [u]_i$, le plus grand mot $v$ tel que $u'v \in L_i$ est aussi le plus grand mot $v$ tel que $uv \in L_i$ et, par suite,

$$\mathrm{pref}(u') = \mathrm{pref}_i(u') = \mathrm{pref}_i(u) = \mathrm{pref}(u).$$

Il existe donc une application pref de $S = \{[u]; u \in F(L)\}$ dans $\mathbb{N}$ telle que, $\forall u \in F(L)$, $\mathrm{pref}([u]) = |\mathrm{pref}(u)|$.

Par exemple, si $L = (abbabc, bababc)$ comme dans l'exemple après la Définition 1.2, $\mathrm{pref}(abc) = \mathrm{pref}_1(abc) = abbabc$ et $\mathrm{pref}([abc]) = 6$. (voir la Fig. 1).

(ii) L'application partielle $\tau : S \times A^* \to S$ définie en $([u], v) \in S \times A^*$ si, et seulement si, $uv \in F(L)$ et telle que, dans ce cas, $\tau([u], v) = [uv]$ est appelée *fonction de transition de progression.* L'application partielle $\varphi$ de $S \times A^*$ dans $\mathbb{Z}$ définie en $([u], v)$ si, et seulement si, $uv \in F(L)$ et telle que, dans ce cas,

$$\varphi([u], v) = \mathrm{pref}([uv]) - \mathrm{pref}([u]) - |v|$$

est appelée *fonction de sortie.* Elle est définie par sa restriction à $S \times A$. $\mathfrak{F}(L) = (S, \tau, \phi)$ est alors un transducteur appelé le *transducteur de progression dans L.* Ce transducteur généralise le transducteur introduit par Crochemore [12] pour un mot.

Par exemple, si $L = (abbabc, bababc)$, comme dans l'exemple après la Définition 1.2,

$$\varphi(3, c) = \mathrm{pref}(7) - \mathrm{pref}(4) - 1 = 3$$

et

$$\varphi(10, a) = \text{pref}(11) - \text{pref}(10) - 1 = |\text{pref}_2(baba)| - |\text{pref}_1(bab)| - 1 = -2.$$

Dans nos exemples, la valeur de $\varphi(s, a)$ ne sera donnée que si elle est non nulle.

Comme $\mathfrak{F}(L)$ dépend de l'ordre des mots de $L$, on supposera dorénavant que le langage $L$ est donné sous la forme d'une suite de mots $(x_1, \ldots, x_t)$. En outre, on supposera aussi que $|x_1| \geq |x_2| \geq \cdots \geq |x_t|$ car cette hypothèse permet d'éliminer facilement les mots redondants.

Si $u \in F(L)$ et $s \in S$ sont tels que $\tau([1], u) = s$, on dit que le *mot* $u$ est *reconnu par l'état* $s$. Le *langage* $\{u \in A^*; \tau([1], u) = s\}$, c'est-à-dire la classe $[u]$ de $u$ modulo $\text{Pd}(\tilde{L})$ est aussi dit *reconnu par l'état* $s$. Si $u$ est le plus grand *mot* reconnu par un état $s$, alors on dit que $u$ *représente l'état* $s$. Par abus de langage, la classe $[u]$ de $u$ modulo $\text{Pd}(\tilde{L})$ est identifiée à l'état $s$ qui la reconnaît.

**1.6. Définition.** (i) $\forall u \in F(L)\backslash\{1\}$, soit $\text{suf}(u) = \text{suf}_L(u)$ le plus grand mot de $\text{Fd}(u)\backslash[u]$. Si $L = (abbabc, bababc)$ comme après la Définition 1.2, $\text{suf}(bababc) = babc$ et $\text{suf}(babc) = 1$. D'après la Proposition 1.4, si $u$ est le mot le plus long de $[u]$, alors $[u] = \text{Fd}(u)\backslash\text{Fd}(\text{suf}(u))$ et, par suite, $\forall v \in [u]$, $\text{suf}(v) = \text{suf}(u)$. Il existe donc une application suf de $S\backslash\{[1]\}$ dans $S$, appelée le *lien suffixe* de $F(L)$, telle que, $\forall u \in F(L)\backslash\{1\}$, $\text{suf}([u]) = [\text{suf}(u)]$. En introduisant un état fictif 0 tel que, $\forall a \in A$, $\tau(0, a) = [1]$, on peut poser $\text{suf}([1]) = 0$.

(ii) Comme le langage $F(L)$ est fini et que tous les états de $S$ sont terminaux, le graphe de l'automate partiel $(S, \tau)$ est sans circuits et, $\forall s \in S$, le maximum des longueurs des chemins d'extrémité terminale $s$ est un entier naturel $\text{rg}(s)$ appelé le *rang* de $s$. Si $u$ est le plus long mot de $[u]$, alors $\text{rg}([u]) = |u|$.

(iii) $\forall u \in F(L)$, soit $\text{part}(u)$ le plus petit entier naturel $i$ de $\{1, \ldots, t\}$ tel que $u \in F(x_i)$ et, si $u$ est le plus grand mot de $[u]$, soit $\text{part}([u]) = \text{part}(u)$. La classe $[u]$ peut contenir des mots $v$ tels que $\text{part}(v) < \text{part}(u)$ comme, par exemple, lorsque $L = (abbcabc, abbabc, ababcb)$ (voir Fig. 5 en la Section 3), $[ababc] = \{ababc, babc, abc\}$ est tel que $\text{part}([ababc]) = 3$ alors que $\text{part}(abc) = 1$.

(iv) Les *états* $[x_1], [x_2], \ldots, [x_t]$ n'ont pas de successeurs dans le graphe de $\mathfrak{F}(L)$ alors que tous les autres états en ont: ils sont dits *ultimes*.

## 2. La taille du transducteur $\mathfrak{F}(L)$

**2.1. Lemme.** *Pour tout langage fini $L$ sans mots redondants, le nombre $e$ d'états de chacun des automates $\mathfrak{M}(\tilde{L})$, $\mathfrak{M}(\tilde{K})$ et $\mathfrak{M}(\tilde{J})$ vérifie*

$$\max\{|x_i|;\ i \in \{1, \ldots, t\}\} + |L| \leq e \leq 2\|L\| - 3|L| + 2$$

*avec* $\|L\| = \sum_{x \in L} |x|$ *et ces bornes sont optimales.*

**Preuve.** (i) Lorsque $L=\{x_1\}$, $\tilde{K}=\tilde{J}=(F(x_1))$ et $\tilde{L}=(L_1)$ avec $L_1=F(x_1)\backslash F(\check{x}_1)$. D'après la Proposition 1.3, $\mathfrak{M}(\tilde{L})$ reconnaît le langage $F(L)=F(x_1)$ et est un quotient de $\mathfrak{M}(\tilde{K})$. Comme $\mathfrak{M}(\tilde{K})$ est l'automate minimal de $F(x_1)$, les automates $\mathfrak{M}(\tilde{L})$ et $\mathfrak{M}(\tilde{K})=\mathfrak{M}(\tilde{J})$ sont isomorphes. En outre, en enlevant l'état qui reconnaît les mots de $A^*\backslash F(x_1)$ de l'automate $\mathfrak{M}(\tilde{J})$ on trouve l'automate partiel minimal de $F(x_1)$ et $e\leqslant 2|x_1|-1$ d'après [12] (voir aussi [7]) d'où $e\leqslant 2\|L\|-3|L|+2$.

(ii) Supposons que, pour l'entier $t$, pour toute suite $L=(x_1,\ldots,x_t)$ de $t$ mots, si $\tilde{J}=(F(x_1),\ldots,F(x_t))$, le nombre $e$ d'états de $\mathfrak{M}(\tilde{J})$ vérifie $e\leqslant 2\|L\|-3|L|+2$ et soient $x_{t+1}\in A^*\backslash F(L)$, $L'=L\cup\{x_{t+1}\}$ et $\tilde{J}'=(F(x_1),\ldots,F(x_t),F(x_{t+1}))$.

D'après la Proposition 1.3 appliquée au cas où $L$ est réduit à un unique mot (voir aussi [12, 5, 6]), $\forall i\in\{1,\ldots,t+1\}$, $\forall u\in F(x_i)$, la classe $[u]_{x_i}$ de $u$ modulo $\mathrm{Pd}(F(x_i))$ est de la forme $\mathrm{Fd}(u)\backslash\mathrm{Fd}(v)$, lorsque $u$ est le plus grand mot de $[u]_{x_i}$ et que $v$ est le plus grand mot $\mathrm{Fd}(u)\backslash[u]_{x_i}$. Comme

$$\mathrm{Pd}(\tilde{J}')=\bigcap_{i\in 1,\ldots,t+1}\mathrm{Pd}(F(x_i))=\mathrm{Pd}(\tilde{J})\cap\mathrm{Pd}(F(x_{t+1})),\quad\forall u\in F(L)$$

(respectivement $u\in F(L')$), la classe $[u]_J$ (respectivement $[u]_{J'}$) de $u$ modulo $\mathrm{Pd}(\tilde{J})$ (respectivement $\mathrm{Pd}(\tilde{J}')$) est aussi de cette forme et $[u]_{J'}=[u]_J\cap[u]_{x_{t+1}}$.

Par suite, $\forall u\in\mathrm{Fg}(L')$, si $k$ (respectivement $k'$, $k''$) est le nombre de classes modulo $\mathrm{Pd}(\tilde{J})$ (respectivement $\mathrm{Pd}(\tilde{J})$, $\mathrm{Pd}(F(x_{t+1}))$) contenues dans $\mathrm{Fd}(u)\backslash\{1\}$, alors $k'\leqslant k+k''$. Il en résulte que le nombre $e'$ d'états de $\mathfrak{M}(\tilde{J}')$ vérifie

$$e'\leqslant e+2|x_{t+1}|-3\leqslant 2(\|L\|+|x_{t+1}|)-3(|L|+1)+2=2\|L'\|-3|L'|+2.$$

L'inégalité $e\leqslant 2\|L\|-3|L|+2$ en résulte par récurrence sur $t=|L|$.

(iii) Si $a_1,\ldots,a_t,b_1,\ldots,b_t,c_1,\ldots,c_t$ sont des lettres de $A$ deux à deux distinctes et si, $\forall i\in\{1,\ldots,t\}$, $x_i=a_ib_i^{k_i}c_i$ avec $k_i>0$ et si $L=\{x_1,\ldots,x_t\}$, $\mathfrak{M}(\tilde{L})$ est isomorphe à $\mathfrak{M}(\tilde{J})$ car, $\forall i\in\{1,\ldots,t\}$, $L_i=\mathrm{Fd}(x_i)\backslash\{1\}$ et $e=2\|L\|-3|L|+2$ ce qui prouve que la borne supérieure peut être atteinte pour $\mathfrak{M}(\tilde{J})$, $\mathfrak{M}(\tilde{K})$ et $\mathfrak{M}(\tilde{L})$.

(iv) Lorsque $L$ ne contient aucun mot redondant, alors $\max\{|x_i|;i\in\{1,\ldots,t\}\}+|L|\leqslant e$. La borne inférieure est atteinte dans le cas suivant: $a_1,a_2,\ldots,a_t$ étant des lettres de $A$ deux à deux distinctes, si $x_i=a_1^{k_i}a_i$ avec $k_i>0$, $\forall i\in\{1,\ldots,t\}$ et si $L=\{x_1,\ldots,x_t\}$, $\forall i\in\{1,\ldots,t\}$, alors $L_i=\mathrm{Fd}(x_i)\backslash\{1\}$ et, par suite, $\mathfrak{M}(\tilde{K})=\mathfrak{M}(\tilde{L})$ et $\mathfrak{M}(\tilde{J})$ cont isomorphes et $e=\max\{|x_i|;i\in\{1,\ldots,t\}\}+|L|$. □

**2.2. Lemme.** *Si $L=\{x_1,\ldots,x_t\}$ et si, $\forall i\in\{1,\ldots,t\}$, $L_i=\mathrm{Fd}(x_i)\backslash F(x_1,\ldots,x_{i-1},\check{x}_i,\ldots,\check{x}_t)$, le nombre d'états $e(L)$ et le nombre de transitions $a(L)$ du transducteur $\mathfrak{F}(L)$ sont tels que*

$$\|L\|\leqslant a(L)\leqslant e(L)+\sum_{i=1}^{t}|L_i|-|L|-1.$$

**Preuve.** (i) Si $U=\{(s,t)\in S\times S;(\exists a\in A)\,\tau(s,a)=t\}$, $(S,U)$ est le graphe de l'automate partiel $(S,\tau)$ et du transducteur $\mathfrak{F}(L)=(S,\pi,\varphi)$.

Si $V$ est l'ensemble des arcs $(s, t)$ de $U$ tels que $\mathrm{rg}(t) = \mathrm{rg}(s)+1$, $(S, V)$ est une arborescence car, si $u$ est le mot le plus long reconnu par l'état $s$, $|u| = \mathrm{rg}(s)$ et, si $a$ est la lettre de $A$ telle que $\tau(s, a) = t$, alors $ua$ est le plus grand mot reconnu par l'état $t$ d'après la Proposition 1.4. Il en résulte que $|V| = e(L) - 1$.

(ii) $\forall (s, t) \in U \setminus V$, soient $u$ le mot le plus long reconnu par $s$, $a$ la lettre de $A$ telle que $\tau(s, a) = t$, $i = \mathrm{part}(t)$ et $v = [\mathrm{pref}(ua)]^{-1} x_i$. Alors $uav \in \mathrm{Fd}(x_i)$, $uav \notin F(x_1, \ldots, x_{i-1})$ puisque $\mathrm{part}(t) = i$, $uav \notin F(\check{x}_i)$ d'après la définition de $\mathrm{pref}_i(ua)$ et $uav \notin F(\check{x}_{i+1}, \ldots, \check{x}_t)$ puisque $\mathrm{part}(t) = i$ et

$$|auv| = |u| + 1 + |v| \leqslant \mathrm{rg}(s) + 1 + |v| < \mathrm{rg}(t) + |v| \leqslant \mathrm{rg}([uav]).$$

Il en résulte que $uav \in L_i$ et $uav \neq x_i$ puisque $|uav| < \mathrm{rg}([uav]) = |x_i|$.

On définit ainsi une application de $U \setminus V$ dans $\bigcup_{i \in 1, \ldots, t} (L_i \setminus \{x_i\})$ et cette application est injective car, si $(s', t')$ est un arc de $U \setminus V$ distinct de $(s, t)$ et si $u'a'v'$ est le mot associé à $(s', t')$, alors $s \neq s'$ implique $u \neq u'$ et $s = s'$ et $t \neq t'$ implique $a \neq a'$ d'où $uav \neq u'a'v'$ dans tous les cas. Il en résulte que $\mathrm{card}(U \setminus V) \leqslant \sum_{i=1}^{t} |L_i| - |L|$ et, d'après (i), que

$$a(L) \leqslant e(L) - 1 + \sum_{i=1}^{t} |L_i| - |L|.$$

(iii) Lorsque $L$ est le langage du (iii) de la démonstration du Lemme 2.1, la borne supérieure

$$e(L) - 1 + \sum_{i=1}^{t} |L_i| - |L| = 3\|L\| - 4|L|$$

est atteinte.

(iv) L'inégalité $\|L\| \leqslant a(L)$ se prouve facilement par récurrence sur $\|L\|$.

En outre, la borne inférieure est atteinte dans le cas où les mots $x_1, \ldots, x_t$ de $L$ sont de la forme $x_i = a_i^{k_i}$ avec $k_i > 0$ et des lettres $a_1, \ldots, a_t$ deux à deux distinctes. □

**2.3. Théorème.** *Pour tout langage fini $L$ sans mots redondants, le nombre $e(L)$ d'états et $a(L)$ de transitions du transducteur $\mathfrak{F}(L)$ vérifient*

$$\max\{|x_i|; i \in \{1, \ldots, t\}\} + |L| \leqslant e(L) \leqslant 2\|L\| - 3|L| + 1$$

*et*

$$|L| \leqslant a(L) \leqslant 3\|L\| - 4|L|.$$

*En outre les bornes données sont optimales.*

**Preuve.** Comme $\mathfrak{F}(L)$ se déduit de $\mathfrak{M}(\tilde{L})$ par la suppression de l'état qui reconnaît les mots de $A^* \setminus F(L)$, $e(L) = e - 1$ et la première inégalité découle du Lemme 2.1. En outre, $\forall i \in \{1, \ldots, t\}$, $L_i \subseteq \mathrm{Fd}(x_i) \setminus \{1\}$ d'où $|L_i| \leqslant |x_i|$ et $\sum_{i=1}^{t} |L_i| \leqslant \|L\|$ d'où $\|L\| \leqslant a(L) \leqslant 3\|L\| - 4|L|$ d'après le Lemme 2.2.

Les exemples donnés dans les preuves des Lemmes 2.1 et 2.2 montrent que les bornes données peuvent être atteintes. □

## 3. Les théorèmes fondamentaux

**3.1. Définition.** Soit $L=\{x_1,\dots,x_t\}$ avec des mots $x_1,\dots,x_{t-1}$ non redondants et $x_t$ éventuellement redondant pour $L$.

(i) Pour toute lettre $a$ de $A$, la suite $(z_0,z_1,\dots,z_s)$ de facteurs droits de $x_t$ telle que $z_0=x_t$ et, $\forall r\in\{1,\dots,s\}$, $z_r=\mathrm{suf}(z_{r-1})$ et $z_ra\notin F(L)$ et $\mathrm{suf}(z_s)a\in F(L)$ lorsque $z_s\neq 1$ est appelée la *suite suffixe de $x_t$ relative à la lettre $a$* (dans $\mathfrak{F}(L)$).

(ii) Soient $x_t'=x_ta$ et $L'=\{x_1,\dots,x_{t-1},x_t'\}$ tels que $L'$ soit sans mots redondants. $\forall i\in\{1,\dots,t-1\}$,

$$L_i'=\mathrm{Fd}(x_i)\backslash F(x_1,\dots,x_{i-1},\check{x}_i,\dots,\check{x}_t')=L_i\backslash\mathrm{Fd}(x_t)$$

et

$$L_t'=\mathrm{Fd}(x_t')\backslash F(L)=\left(\bigcup_{r=0}^{s}[z_r]\cap\mathrm{Fd}(x_t)\right)a.$$

En outre, soient $\tilde{L}=(L_1,\dots,L_t)$ avec $L_t=\emptyset$ lorsque $x_t$ est redondant pour $L$, $\tilde{L}'=(L_1',\dots,L_t')$, $\tilde{L}''=(L_1,\dots,L_{t-1},L_t')$ et, $\forall u\in A^*$, $[u]$ (respectivement $[u]'$, $[u]''$) la classe de $u$ modulo $\mathrm{Pd}(\tilde{L})$ (respectivement $\mathrm{Pd}(\tilde{L}')$, $\mathrm{Pd}(\tilde{L}'')$).

**3.2. Théorème.** *Si $(z_0,z_1,\dots,z_s)$ est la suite suffixe de $x_t$ relative à la lettre $a$ et si $u\in F(L)$, alors $[u]\neq[u]''$ si, et seulement si, $\exists v\in[u]\cap\mathrm{Fg}(z_0,\dots,z_s)$ tel que $|v|<\mathrm{rg}([u])$ et, dans ce cas, $v$ est unique, $[u]''=[u]\backslash\mathrm{Fd}(v)$ et $[v]''=[u]\cap\mathrm{Fd}(v)$.*

**Preuve.** (i) Si $L_t=\emptyset$, $\mathrm{Pd}(\tilde{L}'')=\mathrm{Pd}(\tilde{L})\cap\mathrm{Pd}(L_t')\subseteq\mathrm{Pd}(\tilde{L})$. Si $L_t\neq\emptyset$, $\forall u\in L_t$ et $\forall v\in[u]''$, comme $v^{-1}L_i=u^{-1}L_i=\emptyset$ $\forall i\in\{1,\dots,t-1\}$, nous avons $v\notin F(x_1,\dots,x_{t-1})$. En outre, comme $v^{-1}L_t'=u^{-1}L_t'=\{a\}$,

$$v\in L_t'a^{-1}=\bigcup_{r=0}^{s}[z_r]\cap\mathrm{Fd}(x_t)\quad\text{et}\quad v\notin F(\check{x}_t).$$

Or, $\forall r\in\{1,\dots,s\}$, $[z_r]\subseteq F(x_1,\dots,x_{t-1},\check{x}_t)$ et, par suite, $v\in[z_0]=[x_t]=L_t$ d'où $[u]''\subseteq L_t$. $\mathrm{Pd}(\tilde{L}'')$ sature donc les langages $L_1,\dots,L_t$ et, par suite, $\mathrm{Pd}(\tilde{L}'')\subseteq\mathrm{Pd}(\tilde{L})$.

(ii) S'il existe $r\in\{0,\dots,s\}$ et $v\in[u]\cap\mathrm{Fg}(z_s)$ tels que $|v|<\mathrm{rg}([u])$, il existe aussi $c\in A$ tel que $cv\in[u]$ et, si $w=v^{-1}z_r$,

$$cz_r=cvw\in[v]w\subseteq[vw]=[z_r].$$

Si $r=0$, $z_r=x_t$ et $cvw\notin\mathrm{Fd}(x_t)$ et sinon, comme $z_r$ est le plus grand mot de $\mathrm{Fd}(z_{r-1})\backslash[z_{r-1}]$, $cz_r\notin\mathrm{Fd}(z_{r-1})$ d'où encore $cz_r\notin\mathrm{Fd}(x_t)$ puisque

$$z_{r-1}\in\mathrm{Fd}(z_{r-2})\subseteq\cdots\subseteq\mathrm{Fd}(z_0)=\mathrm{Fd}(x_t)$$

d'après la Proposition 1.4. Il en résulte que $vwa\in\mathrm{Fd}(x_t')\backslash F(L)=L_t'$ et que $cvwa\notin\mathrm{Fd}(x_t')$ d'où $cvwa\notin L_t'$, $cv\notin[v]''$ et $[u]''\neq[u]$.

(iii) Réciproquement, si $[u]\neq[u]''$, comme $[u]''\subseteq[u]$ d'après (i), $\exists v\in[u]$ et $\exists c\in A$ tels que $cv\in[u]$ et $(v,cv)\notin\mathrm{Pd}(\tilde{L}'')$. Alors $(v,cv)\notin\mathrm{Pd}(L_t')$ et $\exists w\in A^*$ tel que $vwa\in L_t'$ et $cvwa\notin L_t'$. Ceci implique que $cvw\notin\mathrm{Fd}(x_t)$ et, comme $L_t'=$

$(\bigcup_{r=0}^{s}[z_r]\cap \mathrm{Fd}(x_t))a$, $\exists r\in\{0,\dots,s\}$ tel que

$$vw\in[z_r]\cap \mathrm{Fd}(x_t) \quad \text{et} \quad cvw\notin[z_r]\cap \mathrm{Fd}(x_t).$$

Si $r>0$, $\exists b\in A$ tel que $bvw\in \mathrm{Fd}(x_t)$ et $b\neq c$ puisque $cvw\notin \mathrm{Fd}(x_t)$ et, comme $cvw\in[v]w\subseteq[vw]=[z_{r-1}]$, $vw=z_r$.

Si $r=0$, $\nexists b\in A$ tel que $bvw\in \mathrm{Fd}(x_t)$ car sinon, comme précédemment, $b\neq c$ et $bvw\notin[vw]$ ce qui implique $vw\notin[z_0]$ contrairement à l'hypothèse. Il en résulte que $vw=z_0=x_t$.

Dans tous les cas, $v\in[u]\cap \mathrm{Fg}(z_s)$ et $|v|<|cv|\leqslant \mathrm{rg}([u])$.

(iv) Soient $v\in[u]$ et $c\in A$ tels que $cv\in[u]$ et $(v,cv)\notin \mathrm{Pd}(L_t')$ et soit $v'\in[u]\cap \mathrm{Fd}(v)$. Comme $v'\in \mathrm{Fd}(v)$, $v^{-1}L_t'\subseteq v'^{-1}L_t'$. $\forall w\in A^*$ tel que $v'wa\in L_t'$, $v'w\in \mathrm{Fd}(x_t)$ et $vw\in[v]w\subseteq[vw]\subseteq F(L)$ d'après la Proposition 1.3 et $v'w\in \mathrm{Fd}(x_t)$ d'après la Proposition 1.4. Comme $v'wa\in L_t'$, $v'wa\notin F(L)$ d'où $vwa\notin F(L)$ et, par suite, $vwa\in L_t'$ et $v'^{-1}L_t'\subseteq v^{-1}L_t'$. Il en résulte que $(v,v')\in \mathrm{Pd}(L_t')$ d'où $(v,v')\in \mathrm{Pd}(\tilde{L}'')$ et, par suite, $[u]\cap \mathrm{Fd}(v)\subseteq[v]''$.

D'après (i), $[v]''\subseteq[v]=[u]$ et, d'après (ii), $\exists w\in A^*$ tel que $vwa\in L_t'$ et $cvwa\notin L_t'$ et, par suite, $\forall u'\in[u]\backslash \mathrm{Fd}(v)$, $u'wa\notin L_t'$ d'où $u'\in[v]''$. Il en résulte que $[v]''=[u]\cap \mathrm{Fd}(v)$.

(v) D'après (iv), il existe au plus un mot $v\in[u]$ et une lettre $c\in A$ tels que $cv\in[u]$ et $(v,cv)\notin \mathrm{Pd}(L_t')$ et, dans ce cas, $\forall u'\in[u]\backslash \mathrm{Fd}(v)$, $cv\in \mathrm{Fd}(u')$ d'où $u'^{-1}L_t'\subseteq(cv)^{-1}L_t'=\emptyset=u^{-1}L_t'$, $(u,u')\in \mathrm{Pd}(L_t')$, $(u,u')\in \mathrm{Pd}(L'')$ et, par suite, $[u]''=[u]\backslash \mathrm{Fd}(v)$. □

**Exemple.** Si $x_1=abbcabc$, $x_2=abbabc$ et $x_3=ababc$ et si $L=\{x_1,x_2,x_3\}$, l'automate $\mathfrak{F}(L)$ est représenté par le graphe de la Fig. 3. Dans ce cas, $L_1=\{abbcabc, bbcabc, bcabc, cabc, abc\}$, $L_2=\{abbabc, bbabc, babc\}$ et $L_3=\{ababc\}$ et la suite suffixe de $x_3$ relative à la lettre $b$ est $(z_0,z_1,z_2,z_3)$ avec $z_0=x_3$, $z_1=babc$, $z_2=abc$ et $z_3=bc$.

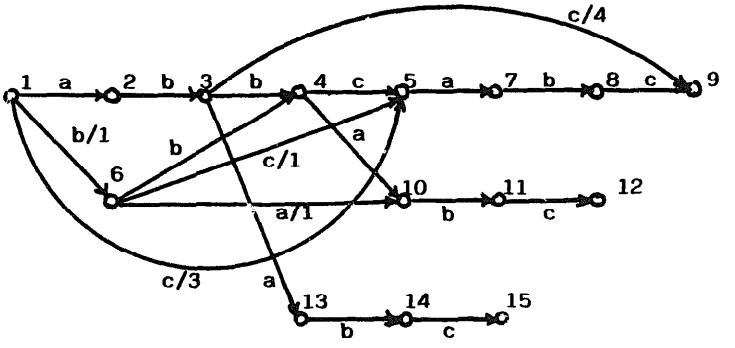


Fig. 3.

Si $x_3'=x_3b=ababcb$, $L_3'=\{ababcb, babcb, abcb, bcb, cb\}$ et l'automate partiel $\mathfrak{M}^*(\tilde{L}'')$ obtenu en supprimant dans $\mathfrak{M}(\tilde{L}'')$ l'état $A^*\backslash F(L')$ est représenté par la Fig. 4.

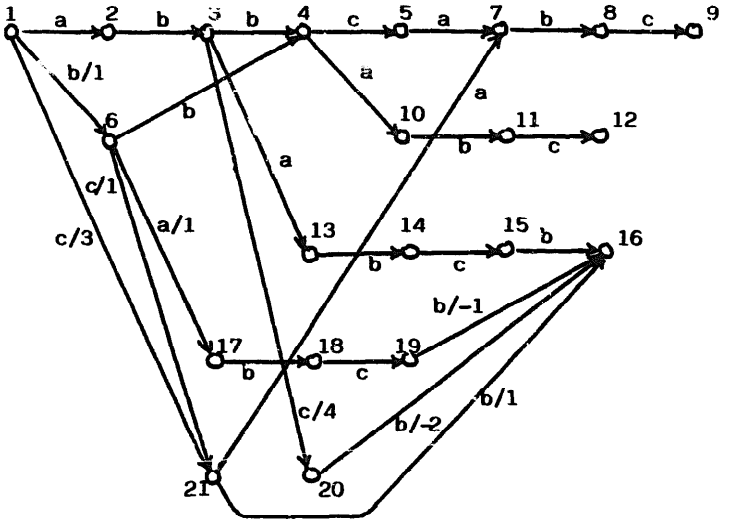


Fig. 4.

Les états $10=[abba]$, $11=[abbab]$, $12=[abbabc]$, $9=[abbcabc]$ et $5=[abbc]$ de la Fig. 3 sont respectivement décomposés en les états $10=[abba]''=\{abba, bba\}$ et $17=[ba]''=\{ba\}$, $11=[abbab]''=\{abbab, bbab\}$ et $18=[bab]''=\{bab\}$, $12=[abbabc]''=\{abbabc, bbabc\}$ et $19=[babc]''=\{babc\}$, $9=[abbcabc]''=\{abbcabc, bbcabc, bcabc, cabc\}$ et $20=[abc]''=\{abc\}$, $5=[abbc]''=\{abbc, bbc\}$ et $21=[bc]''=\{bc, c\}$ de la Fig. 4.

**3.3. Définition.** La suite $(z_0, z_1, \ldots, z_r)$ de facteurs droits de $x_t$ telle que $z_0=x_t$ et $\forall i\in\{1,\ldots,r\}$, $z_i=\mathrm{suf}(z_{i-1})$ dans $\mathfrak{F}(L)$ et $z_i\in\bigcup_{j=1}^{t+1}L_j$ et $\mathrm{suf}(z_r)\notin\bigcup_{j=1}^{t-1}L_j$ est appelée la *suite suffixe ultime de* $x_t$.

Pour toute lettre $a$ de $A$, $(z_0,\ldots,z_r)$ est une sous-suite de la suite suffixe $(z_0,\ldots,z_s)$ de $x_t$ relative à $a$. Il existe alors des entiers $k_1>k_2>\cdots>k_r$ de $\{1,\ldots,t-1\}$ tels que, $\forall i\in\{1,\ldots,r\}$, $z_i$ soit le plus grand mot de $L''_{k_i}=L_{k_i}\cap\mathrm{Fd}(x_t)$ et $[z_i]''=L''_{k_i}$ car, $\forall u\in L''_{k_i}$, $u^{-1}L_{k_i}=\{1\}$, $u^{-1}L'_t=\{a\}$ et, $\forall j\in\{1,\ldots,t-1\}\backslash\{i\}$, $u^{-1}L_j=\emptyset$ et, $\forall u'\in L'_{k_i}$, $u'^{-1}L'_t=\emptyset$.

**3.4. Théorème.** *Si* $(z_0, z_1, \ldots, z_r)$ *est la suite suffixe ultime de* $x_t$ *et si* $u\in F(L)$, *alors* $[u]'\neq[u]''$ *si, et seulement si,* $\exists v\in[u]'\cap\mathrm{Fg}(z_1,\ldots,z_r)$ *tel que* $|v|<\mathrm{rg}([u]')$ *et, dans ce cas, il existe un facteur droit* $w$ *de* $x_t$ *et deux entiers* $k\leqslant l$ *dans* $\{1,\ldots,r\}$ *tels que, lorsque* $u$ *est le plus grand mot de* $[u]'$,

$$[u]'=[u]''\cup\left(\bigcup_{i=k}^{l}[z_iw^{-1}]''\right).$$

**Preuve.** (i) $\mathrm{Pd}(\tilde{L}'')=\mathrm{Pd}(L_1)\cap\cdots\cap\mathrm{Pd}(L_{t-1})\cap\mathrm{Pd}(L'_t)$ sature les langages $L_1,\ldots,L_{t-1}$ et $L'_t$ et, par suite, aussi $L'_ta^{-1}=\bigcup_{i=0}^{s}[z_i]''$ avec $[z_0]''=L_t$, $[z_1]''=$

$L''_{k_1}, \ldots, [z_r]'' = L''_{k_r}$. Alors, $\mathrm{Pd}(\tilde{L}'')$ sature aussi les langages $L''_i = L_i \cap L'_i a^{-1}$ et $L'_i = L_i \backslash L''_i$ $\forall i \in \{1, \ldots, t-1\}$. Il en résulte que $\mathrm{Pd}(\tilde{L}'') \subseteq \mathrm{Pd}(\tilde{L}')$.

(ii) S'il existe $v \in [u]' \cap \mathrm{Fg}(z_1, \ldots, z_r)$ tel que $|v| < \mathrm{rg}([u]')$, $\exists c \in A$ tel que $cv \in [u]'$, $\exists i \in \{1, \ldots, r\}$ et $\exists w \in \mathrm{Fd}(x_t)$ tels que $vw = z_i$. Alors $vw$ est le plus grand mot de $L''_{k_i}$ et $cvw \notin L_{k_i}$ d'où $(cv, v) \in \mathrm{Pd}(L_{k_i})$ et $(cv, v) \notin \mathrm{Pd}(\tilde{L}'')$ ce qui prouve que $[u]' \neq [u]''$.

(iii) Si $[u]'$ contient des mots $v$ et $cv$ avec $c \in A$ et tels que $(cv, v) \notin \mathrm{Pd}(L'')$, il existe $i \in \{1, \ldots, t-1\}$ tel que $(cv, v) \in \mathrm{Pd}(L'_i)$ et $(cv, v) \notin \mathrm{Pd}(L_i)$. Comme $\mathrm{Pd}(L'_i) \cap \mathrm{Pd}(L''_i)$ sature $L_i = L'_i \cup L''_i$, $\mathrm{Pd}(L'_i) \cap \mathrm{Pd}(L''_i) \subseteq \mathrm{Pd}(L_i)$ et, par suite, $(cv, v) \notin \mathrm{Pd}(L''_i)$. Il existe donc $w \in A^*$ tel que $cvw \in L''_i \subseteq L_i$ et $vw \notin L''_i$. Comme $cvwa \in L'_i$ et que $(cv, v) \in \mathrm{Pd}(L'_i)$, $vwa \in L'_i$. Il existe donc $j < i$ tel que $vw$ soit le plus grand mot de $L''_j$ et, par suite, $vw \in \{z_1, \ldots, z_r\}$.

(iv) Si $[u]' \neq [u]''$ et si $u$ est le représentant de $[u]'$, comme $[u]'' \subseteq [u]'$ d'après (i), $u$ est aussi le représentant de $[u]''$ et $[u]' \backslash [u]'' \neq \emptyset$. Si $v$ est le plus grand mot de $[u]' \backslash [u]''$, $\exists c \in A$ tel que $cv$ soit le plus petit mot de $[u]''$ et alors $(cv, v) \in \mathrm{Pd}(\tilde{L}')$ et $(cv, v) \notin \mathrm{Pd}(\tilde{L}'')$. Il en résulte que $v \in [u]' \cap \mathrm{Fg}(z_1, \ldots, z_r)$ d'après (iii) avec $|v| < |cv| \leq \mathrm{rg}([u]')$.

(v) Si $[u]' \neq [u]''$, il existe, d'après (iv), un plus petit entier $k$ (respectivement plus grand entier $l$) de $\{1, \ldots, r\}$ et $w \in \mathrm{Fd}(x_t)$ (respectivement $w' \in \mathrm{Fd}(x_t)$) tel que $z_k w^{-1} \in [u]'$ (respectivement $z_l w'^{-1} \in [u]'$). Comme $z_k w^{-1} wa = z_k a \in L'_t$ et $(z_k w^{-1}, z_l w'^{-1}) \in \mathrm{Pd}(L'_t)$, $z_l w'^{-1} wa \in L'_t$ et, par suite, $w' = w$. Alors, $\forall i \in \{k, \ldots, l\}$, $z_i w^{-1} \in \mathrm{Fd}(z_k w^{-1}) \backslash \mathrm{Fd}(z_l w^{-1}) \subseteq [u]'$ d'après la Proposition 1.4 d'où $[u]'' \cup (\bigcup_{i=k}^{l} [z_i w^{-1}]'') \subseteq [u]'$ d'après (i). Cette inclusion ne peut être stricte d'après (iii) d'où $[u]' = [u]'' \cup (\bigcup_{i=k}^{l} [z_i w^{-1}]'')$. □

**Exemple.** Lorsque $L' = (x_1, x_2, x'_3)$ avec $x_1 = abbcabc$, $x_2 = abbabc$ et $x'_3 = ababcb$ comme dans l'exemple précédent, l'automate partiel $\tilde{\mathfrak{S}}(L')$ est représenté par la Fig. 5. Alors $L''_1 = \{abc\}$ et $L''_2 = \{babc\}$ et la suite suffixe ultime de $x_3$ est $(z_0, z_1, z_2)$ avec $z_0 = x_3$, $z_1 = babc$ et $z_2 = abc$. Les états 13, 14 et 15 de la Fig. 5 sont respectivement la réunion des classes $13 = [aba]'' = \{abc\}$ et $17 = [ba]'' = \{ba\}$, $14 = [abab]'' = \{abab\}$ et $18 = [bab]'' = \{bab\}$ et $15 = [ababc]'' = \{ababc\}$, $19 = [babc]'' = \{babc\}$ et $20 = [abc]'' = \{abc\}$ de la Fig. 4.

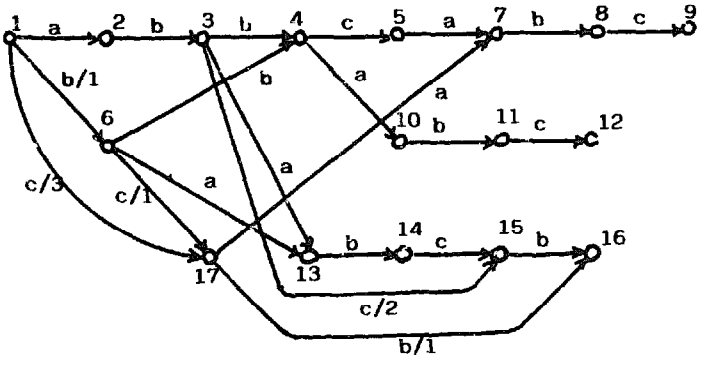


Fig. 5.

## 4. Les procédures de base

**4.1. Définition.** $\forall u \in F(L)\backslash\{1\}$, soit $\text{avs}(u) = |\text{pref}(\text{suf}(u))| - |\text{suf}(u)|$ et, en outre, soit $\text{avs}(1) = 0$. Si $\text{suf}(u) \neq 1$ et si $i = \text{part}([\text{suf}(u)])$, $\text{avs}(u)$ est le nombre de lettres du mot $x_i$ de $L$ situées avant la première occurrence de $\text{suf}(u)$ comme facteur de $x_i$.

$\forall v \in [u]$, $\text{avs}(v) = \text{avs}(u)$ puisque $\text{suf}(v) = \text{suf}(u)$. Il existe donc une application avs de $S$ dans $\mathbb{N}$ telle que, $\forall u \in F(L)$,

$$\text{avs}([u]) = \text{avs}(u) = \text{pref}(\text{suf}([u])) - |\text{suf}(u)|.$$

La fonction avs a été introduite en [20] où elle était notée isuf.

**4.2. Lemme.** *Pour tout* $(s, a) \in S \times A$ *tel que* $\tau(s, a) \neq \emptyset$ *et* $\tau(\text{suf}(s), a) \neq \tau(s, a)$, $\text{avs}(\tau(s, a)) = \text{avs}(s) + \varphi(\text{suf}(s), a)$.

**Preuve.** Si $u$ est le plus petit mot reconnu par $s$, il existe $c \in A$ tel que $u = c\,\text{suf}(u)$. Les mots $ua = c\,\text{suf}(u)a$ et $\text{suf}(u)a$ sont respectivement reconnus par les états $\tau(s, a)$ et $\tau(\text{suf}(s), a)$. Comme, par hypothèse, $\tau(\text{suf}(s), a) \neq \tau(s, a)$, $ua$ est le plus petit mot reconnu par $\tau(s, a)$ et $\text{suf}(ua) = \text{suf}(u)a$ d'où $\text{suf}(\tau(s, a)) = \tau(\text{suf}(s), a)$. Comme $\text{avs}(s) = \text{pref}(\text{suf}(s)) - |\text{suf}(u)|$ et que $\text{avs}(\tau(s, a)) = \text{pref}(\text{suf}(\tau(s, a))) - |\text{suf}(u) \cdot a|$, $\text{avs}(\tau(s, a)) = \text{avs}(s) + \varphi(\text{suf}(s), a)$. □

**4.3. Définition.** (i) Pour tout état $s$ de $\mathfrak{F}(L)$, soit

$$\Delta(s) = \{t ; \text{suf}(t) = \text{suf}(s) \;\&\; \text{avs}(t) = \text{avs}(s)\}$$
$$= \text{suf}^{-1}(\text{suf}(s)) \cap \text{avs}^{-1}(\text{avs}(s)).$$

(ii) Soit $T = \{t_1, \ldots, t_k\}$ une section transversale des classes $\Delta(t)$ contenues dans $\text{suf}^{-1}(s)$ telle que

$$\text{avs}(t_1) < \text{avs}(t_2) < \cdots < \text{avs}(t_k).$$

Les classes $\Delta(t_1), \ldots, \Delta(t_k)$ forment alors une partition de $\text{suf}^{-1}(s)$.

(iii) Soit lis la permutation de $T$ telle que, $\forall i \in \{1, \ldots, k-1\}$, $\text{lis}(t_i) = t_{i+1}$ et $\text{lis}(t_k) = t_1$ et soit alis la permutation inverse de lis. $\forall i \in \{1, \ldots, k\}$ on prolonge l'application lis à $\Delta(t_i)\backslash\{t_i\}$ en posant, $\forall t \in \Delta(t_i)\backslash\{t_i\}$, $\text{lis}(t) = -t_i$.

Soit $\delta$ une application de $S$ dans $S \cup \{0\}$ telle que, $\forall t \in S$, la restriction de $\delta$ à $\Delta(t)$ et $\Delta(t) \cup \{0\}$ soit injective et chaîne les éléments de $\Delta(t)$.

(iv) L'ensemble $\text{suf}^{-1}(s)$ muni de la restriction de lis à $\{t_1, \ldots, t_{k-1}\}$ et de $\delta$ est alors un arbre binaire noté $\mathfrak{B}(s)$.

(v) La racine $t_1$ de $\mathfrak{B}(s)$ est encore appelée la *racine* de $\text{suf}^{-1}(s)$.

Soit rac l'application de $S$ dans $S \cup \{0\}$ telle que, $\forall s \in S$ tel que $\text{suf}^{-1}(s) \neq \emptyset$, $\text{rac}(s)$ soit la racine de $\text{suf}^{-1}(s)$ et, $\forall s \in S$ tel que $\text{suf}^{-1}(s) = \emptyset$, $\text{rac}(s) = 0$. (Voir Fig. 6.)

**4.4. La procédure SITUE.** Sa fonction est de donner tous les paramètres qui situent un état $u$ dans l'arbre $\mathfrak{B}(\text{suf}(u))$: $\text{ul} = \text{lis}(u)$, $\text{ud} = \delta(u)$; si $\text{ul} > 0$, alors $\text{ual} = \text{alis}(u)$

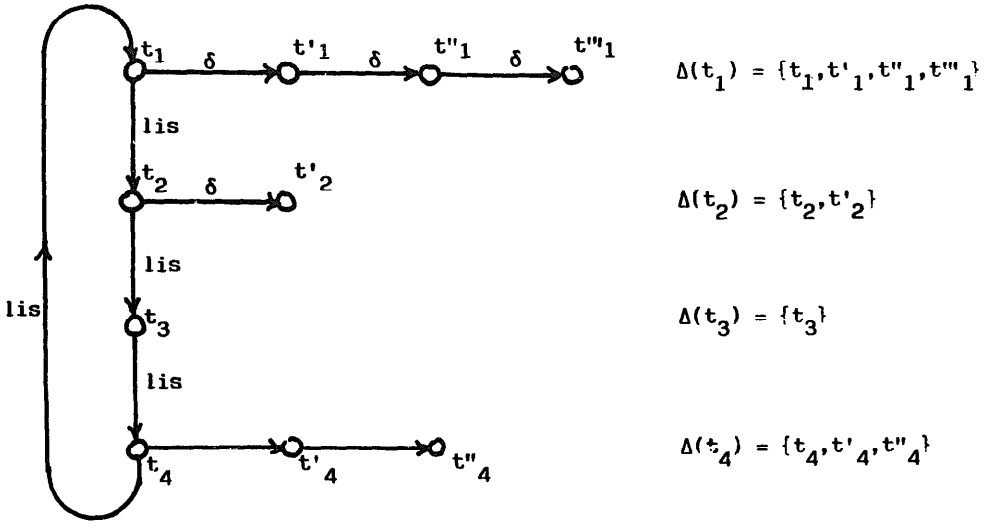


Fig. 6.

et sinon, uad l'état de $\Delta(s)$ tel que $\delta(\text{uad}) = u$ et ur qui est égal à suf($u$) lorsque $u$ est la racine de $\mathfrak{B}$(suf($u$)) et à 0 sinon.

**Procedure** SITUE ($u$:integer);
**begin** ul := lis($u$); ud := $\delta$($u$);
  **if** ul > 0
  **then** ual := alis($u$)
  **else begin** uad := −ul; **while** $\delta$(uad) <> $u$ **do** uad := $\delta$(uad) **end**;
  **if** rac(suf($u$)) = $u$ **then** ur := suf($u$) **else** ur := 0;
**end.**

**4.5. La procédure** PLACE. Sa fonction est de placer un état $u$ dans un arbre $\mathfrak{B}(s)$ a une place prédéterminée par les paramètres ul, ual, ud, uad et ur.

**Procedure** PLACE($u$, ul, ual, ud, uad, ur:integer);
**begin** $\delta$($u$) := ud; **if** ur > 0 **then** rac(ur) := $u$;
  **if** ul > 0
  **then begin** $s$ := ud; **while** $s$ > 0 **do begin** lis($s$) := −$u$; $s$ := $\delta$($s$) **end**;
    **if** lis(ul) = ul **then begin** lis($u$) := $u$; alis($u$) := $u$ **end**
    **else begin** lis(ual) := $u$; lis($u$) := ul; alis(ul) := $u$; alis($u$) := ual **end end**
  **else begin** $\delta$(uad) := $u$; lis($u$) := ul **end**;
**end.**

**4.6. La procédure** INSERE. Sa fonction est d'abord de trouver une place à un état $u$ dans l'arbre $\mathfrak{B}$(suf($u$)) en fonction de la valeur de avs($u$) et ensuite d'insérer $u$ à cette place.

```
Procedure INSERE(u:integer);
begin su:=suf(u); h:=avs(u); ur:=rac(su); s:=ur;
  if ur=0
  then begin rac(su):=u; lis(u):=u; alis(u):=u; δ(u):=0 end
  else
    if h<avs(ur)
    then
    begin
      rac(su):=u:s1:=alis(ur); lis(s1):=u; lis(u):=ur;
      alis(ur):=u; alis(u):=s1; δ(u):=0;
    end
    else
    begin
      if h>=avs(alis(ur))
      then s:=alis(ur)
      else while avs(s)<h do s:=lis(s)
      if avs(s)=h
      then begin δ(u):=δ(s); δ(s):=u; lis(u):=-s end
      else
      begin
        s0:=alis(s); lis(s0):=u; lis(u):=s;
        alis(s):=u; alis(u):=s0; δ(u):=0;
      end;
    end;
end
```

**4.7. La procédure ENLÈVE.** Elle permet d'éliminer un état donné par ses paramètres.

```
Procedure ENLEVE(ul, ual, ud, uad, ur:integer)
begin
  if ur>0
  then if ud>0 then rac(ur):=ud
  else if lis(ul)=ul then rac(ur):=0 else rac(ur):=ul;
  if ul>0
  then
    if ud>0
    then
      if lis(ul)=ul
      then begin lis(ud):=ud; alis(ud):=ud end
      else begin lis(ual):=ud; lis(ud):=ul; alis(ul):=ud; alis(ud):=ual end;
    else begin lis(ual):=ul; alis(ul):=ual end
  else δ(uad):=ud;
end.
```

**4.8. La procédure ECHANGE.** Elle permute les positions dans les arbres binaires de deux états donnés. La variable cplet exprime que le transfert est partiel lorsque cplet = 0 et complet lorsque cplet = 1 (voir Définition 6.4).

```
Procedure ECHANGE(u, v: integer);
begin SITUE(u); vl := ul; val := ual; vd := ud; vad := uad;
  vr := ur; SITUE(v);
  if cplet = 0
  then PLACE(v, vl, val, vd, vad, vr)
  else ENLEVE(vl, val, vd, vad, vr);
  PLACE(u, ul, ual, ud, uad, ur);
end.
```

**4.9. Remarque.** Les procédures précédentes constituent avec les procédures PARTAGE et DECPARTAGE une couche de base qui peut être remplacée par une autre sans toucher aux autres procédures. Dans une première version chaque classe $\text{suf}^{-1}(x)$ était représentée par une liste doublement chaînée.

## 5. La duplication d'un état

**5.1. Définition.** (i) Pour tout état $s$ tel que $\text{avs}(s) > \text{pref}(\text{suf}(s)) - \text{rg}(\text{suf}(s))$, le *lien suffixe* de $s$ est dit *sécant*. Comme $\text{avs}([u]) = \text{pref}(\text{suf}([u])) - |\text{suf}(u)|$ si $u$ représente $s$, cette condition est équivalente à la condition $|\text{suf}(u)| < \text{rg}(\text{suf}(s))$ qui exprime que $\text{suf}(u)$ n'est pas le plus grand mot reconnu par $[\text{suf}(u)]$.

(ii) $\forall u \in F(L)\backslash\{1\}$ tel que $|\text{suf}(u)| < \text{rg}([\text{suf}(u)])$, l'état $[v]$ qui reconnaît le plus grand facteur gauche $v$ de $\text{suf}(u)$ tel que $|v| = \text{rg}([v])$ est appelé le *point d'éclatement de l'état* $s = [u]$. $\forall u \in F(L)\backslash\{1\}$ tel que $|\text{suf}(u)| = \text{rg}([\text{suf}(u)])$ et pour $u = 1$, on dit que *l'état* $s = [u]$ *n'admet pas de point d'éclatement.*

**5.2. Lemme.** *Si $s$ est un état de $\mathfrak{F}(L)$ qui admet un point d'éclatement, tous les descendants de $s$ dans $\mathfrak{F}(L)$ admettent le même point d'éclatement que $s$ et, pour tout mot $w$ tel que $\tau(s, w) \neq \emptyset$, $\text{suf}(\tau(s, w)) = \tau(\text{suf}(s), w)$.*

**Preuve.** (i) Soit $s$ un état de $\mathfrak{F}(L)$ qui admet un point d'éclatement $r$ et soit $u$ le plus petit mot reconnu par $s$. $\forall a \in A$ tel que $t = \tau(s, a) \neq \emptyset$, $\exists b, c \in A$, $b \neq c$, tels que $u = b\,\text{suf}(u)$ et $c\,\text{suf}(u) \in [\text{suf}(u)]$ d'après la Proposition 1.4. Alors $ua$ est reconnu par $t = \tau(s, a)$ d'où $ua \in F(L)$ et, comme $\text{suf}(u) \in \text{Fd}(u)$, $\text{suf}(u)a \in \text{Fd}(ua)\backslash F(L)$ et $c\,\text{suf}(u)a \in [\text{suf}(u)]a \subseteq [\text{suf}(u)a]$. Comme $|ua| = |c\,\text{suf}(u)a|$, $c\,\text{suf}(u)a \notin [ua]$ ce qui implique $\text{suf}(u)a \notin [ua]$ et, par suite, $\text{suf}(ua) = \text{suf}(u)a$. $\text{suf}(ua)$ admet donc le même plus grand facteur gauche de $v$ tel que $|v| = \text{rg}([v])$ et $t$ admet le même point d'éclatement $r = [v]$ que $s$. En outre, $\text{suf}(ua) = \text{suf}(u)a$ implique $\text{suf}(\tau(s, a)) = \tau(\text{suf}(s), a)$.

(ii) Il résulte de (i) par récurrence sur $|w|$ que, si $\tau(s, w) \neq \emptyset$, alors $\tau(s, w)$ admet le même point d'éclatement que $s$ et $\text{suf}(\tau(s, w)) = \tau(\text{suf}(s), w)$. □

**5.3. La procédure POINTDECLAT.** (i) Afin de pouvoir modifier facilement le point d'éclatement d'un état lors de la construction de $\mathfrak{F}(L)$, l'application partielle de $S$ dans $S$ qui associe, à chaque état $s$ de $S$, son point d'éclatement lorsqu'il existe est décomposée en le produit de deux applications pt et eclat telles que si $s$ est un état dont aucun prédécesseur n'admet de point d'éclatement et si $T$ est l'ensemble des descendants de $s$ dans $\mathfrak{F}(L)$ qui admettent un point d'éclatement $r$, alors $\text{eclat}^{-1}(T)$ est réduit à un élément unique $e$ ce qui permet de changer $r$ en un nouveau point d'éclatement $r'$ en remplaçant $\text{pt}(e) = r$ par $\text{pt}(e) = r'$ et de supprimer le point d'éclatement de $s$ en posant $\text{eclat}(s) = 0$.

(ii) La procédure POINTDECLAT$(s, t, c)$ où $s$, $t \in S$ et $c \in A$ vérifient $\tau(s, c) = t$, affecte le point d'éclatement de $s$ à $t$ s'il existe, teste si $\text{suf}(s)$ n'est pas le point d'éclatement de $t$ et, dans ce cas, l'affecte à $t$ et sinon, pose $\text{eclat}(t) = 0$.

```
Procedure POINTDECLAT(s, t:integer; c:char);
begin
  if eclat(s)>0
  then eclat(t):=eclat(s)
  else
  begin s1:=suf(s);
    if rg(τ(s1, c))>rg(s1)+1
    then begin e:=e+1; eclat(t):=e; pt(e):=s1 end
    else eclat(t):=0;
  end;
end.
```

**5.4. Définition.** (i) $\forall s \in S$ et $\forall a \in A$, la suite $(s_0, s_1, \ldots, s_k)$ d'états de $S$ telle que $s_0 = s$, $\forall i \in \{1, \ldots, k\}$, $s_i = \text{suf}(s_{i-1})$ et $\tau(s_i, a) = \tau(s, a)$ et $\tau(\text{suf}(s_k), a) \neq \tau(s, a)$ est appelée la *suite suffixe de s relative à la lettre a.* $\text{suf}(s_k)$ est appelé la *borne de la suite suffixe* $(s_0, \ldots, s_k)$. Lorsque $s_k = 1$, c'est l'état fictif 0. Cette définition s'applique au cas particulier où $\tau(s, a) = \emptyset$, c'est-à-dire lorsque $\tau$ n'est pas défini en $(s, a)$.

Par exemple, si $L = \{abbcabc, abbabc, ababc\}$ comme dans l'exemple après le Théorème 3.2 (Fig. 3), la suite suffixe de $s = 15$ relative à $b$ est (15, 12, 9, 5) et 1 est sa borne.

(ii) $\forall s \in S$, soit $\text{dde}(s)$ le demi-degré extérieur de $s$ dans le graphe de $\mathfrak{F}(L)$, c'est-à-dire le cardinal de l'ensemble $\{a \in A; \tau(s, a) \neq \emptyset\}$.

(iii) $\forall s \in S \backslash \{[1]\}$, il existe un prédécesseur $t$ du sommet $s$ dont le rang est le plus petit et ce prédécesseur est unique d'après la Proposition 1.4. Ce prédécesseur $t$ est appelé le *dernier prédécesseur de s* et est noté $\text{dpred}(s)$.

(iv) $\forall (r, b) \in S \times A$ tel que $\tau(r, b) \neq \emptyset$, il existe $r_0 \in S$ tel que $\text{suf}(r_0) = r$ et $\tau(r_0, b) = \tau(r, b)$ si, et seulement si, $\text{rg}(\tau(r, b)) > \text{rg}(r) + 1$ car, si $u$ est le mot qui représente

$r$, $ub$ représente $\tau(r, b)$ si, et seulement si, $\mathrm{rg}(\tau(r, b)) = \mathrm{rg}(r) + 1$. En outre, si $r_0$ existe et si $u_0$ est le mot le plus court reconnu par $r_0$, $u_0 b$ est reconnu par $\tau(r, b)$ et $\exists c \in$ tel que $u_0 b = cub$. Comme la lettre $c$ est unique d'après la Proposition 1.4, l'état $r_0$ l'est aussi. Soit $\psi$ l'application partielle de $S \times A$ dans $S$ définie en $(r, b)$ si, et seulement si, $\mathrm{rg}(\tau(r, b)) > \mathrm{rg}(r) + 1$ et telle que, dans ce cas, $r_0 = \psi(r, b)$ vérifie $\mathrm{suf}(r_0) = r$ et $\tau(r_0, b) = \tau(r, b)$.

**5.5. Définition.** Si $(r, b) \in S \times A$ est tel que $d = \tau(r, b) \neq \emptyset$ et $\mathrm{rg}(\tau(r, b)) > \mathrm{rg}(r) + 1$ et si $u$ et $v$ sont les mots qui représentent respectivement $d$ et $r$, alors $vb \in [v]b \subseteq [vb] = [u]$ et $|vb| = \mathrm{rg}(r) + 1 < \mathrm{rg}(d) = |u|$ et l'éclatement de $d$ en deux états qui reconnaissent respectivement les ensembles de mots $[u] \backslash \mathrm{Fd}(vb)$ et $[u] \cap \mathrm{Fd}(vb)$ est appelé la *duplication de d selon* $(r, b)$.

*L'état d'* qui reconnaît les mots de $[u] \cap \mathrm{Fd}(vb)$ est appelé le *dupliqué de d selon* $(r, b)$. On garde la notation $d$ pour l'état qui reconnaît les mots de $[u] \backslash \mathrm{Fd}(vb)$ après la duplication.

La duplication de $d$ selon $(r, b)$ transforme un automate partiel en un autre qui reconnaît les mêmes mots.

Dans le schéma de la Fig. 7, les classes $[u]$ sont représentées par des rectangles et on suppose que les mots de la classe sont placés les uns en-dessous des autres par longueurs décroissantes.

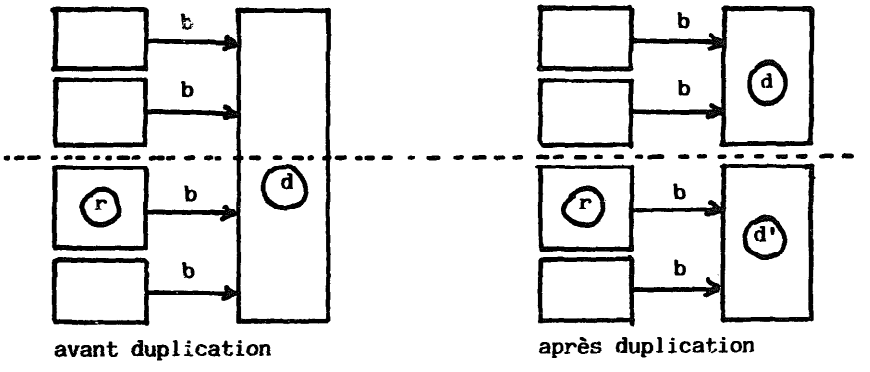


Fig. 7.

**5.6. La procédure DUPLIQUE.** (i) Pour réaliser la duplication de $d$ selon $(r, b)$ il faut:
- remplacer $\tau(s, b) = d$ par $\tau(s, b) = d'$ pour tout état $s$ de la suite suffixe de $r$ relative à $b$;
- transmettre à $d'$ les transitions de $d$ et les valeurs associées de $\varphi$;
- transmettre à $d'$ les attributs $\mathrm{pref}(d)$, $\mathrm{suf}(d)$, $\mathrm{avs}(d)$, $\mathrm{dpred}(d)$, $\mathrm{dde}(d)$ et $\mathrm{eclat}(d)$;
- remplacer $d$ par $d'$ dans l'arbre binaire $\mathfrak{B}(\mathrm{suf}(d))$;
- poser $\mathrm{rg}(d') = \mathrm{rg}(r) + 1$;
- redéfinir $\mathrm{suf}(d) = d'$, $\mathrm{avs}(d) = \mathrm{pref}(d') - \mathrm{rg}(r) - 1$, et $\mathrm{eclat}(d) = 0$ et $\mathrm{dpred}(d) = \psi(r, b)$;
- modifier $\psi$ en conséquence;

- partager $\text{suf}^{-1}(d)$ entre $d$ et $d'$ en appelant la procédure PARTAGE donnée en Définition 5.8.

(ii) La procédure qui suit réalise ces fonctions en prenant comme paramètres $r$ et l'indice $m$ de la lettre $b$ dans le mot courant $x_t$ de $L$ et en créant un nouvel état $q$ pour $d'$. Un tableau let[1, ..., carda] contient les lettres de l'alphabet $A$.

```
Procedure DUPLIQUE(r, m : interger);
begin b := x(m); d := τ(r, b); q := q+1; τ(r, b) := q; pref(q) := pref(d)
  rg(q) := rg(r)+1; avs(q) := avs(d); part(q) := part(d); dde(q) := dde(d);
  eclat(q) := eclat(d); dpred(q) := dpred(d); dpred(d) := ψ(r, b); ψ(r, b) := 0;
  avs(d) := pref(q)−rg(q); SITUE(d); PLACE(q, ul, ual, ud, uad, ur); PARTAGE;
  suf(q) := suf(d); suf(d) := q;
  for k := 1 to carda do
  begin c := let(k); s := τ(d, c);
    if s > 0
    then
    begin τ(q, c) := s; φ(q, c) := φ(d, c); ψ(q, c) := d;
      if ψ(suf(q), c) = d then ψ(suf(q), c) := q;
      if dpred(s) = d then dpred(s) := q;
    end
    else begin τ(q, c) := 0; ψ(q, c) := 0 end;
  end;
  s := suf(r); while τ(s, b) = d do begin τ(s, b) := q; s := suf(s) end;
end.
```

**5.7. Lemme.** *Si $T$ est l'ensemble des états qui admettent $r$ comme point d'éclatement avant la duplication de $d$ selon $(r, b)$ et si $d'$ est le dupliqué de $d$ selon $(r, b)$, alors après la duplication:*

(i) $\text{suf}^{-1}(d)$ *est partagé* $\text{suf}^{-1}(d') = \{s \in \text{suf}^{-1}(d); \text{avs}(s) \geqslant \text{pref}(d) - \text{rg}(r) - 1\}$ *et* $\text{suf}^{-1}(d)/\text{suf}^{-1}(d')$;

(ii) *les états de* $T \cap \text{suf}^{-1}(d)$ *n'admettent plus de point d'éclatement*;

(iii) *le point d'éclatement de tous les états de* $T \setminus \text{suf}^{-1}(d)$ *est changé en* $d'$.

(iv) *aucun autre point d'éclatement n'est créé ou modifié.*

**Preuve.** (i): Si $u$ et $v$ sont les mots qui représentent respectivement $d$ et $r$, $d'$ reconnaît les mots de $[u] \cap \text{Fd}(vb)$. $\forall s \in \text{suf}^{-1}(d)$, si $w$ représente $s$, alors $s \in \text{suf}^{-1}(d')$ si, et seulement si, $\text{suf}(w) \in [u] \cap \text{Fd}(vb)$, c'est-à-dire si, et seulement si,

$$\text{avs}(s) = \text{pref}(d) - |\text{suf}(w)| \geqslant \text{pref}(d) - \text{rg}(r) - 1.$$

(ii): $\forall s \in T \cap \text{suf}^{-1}(d)$, si $w$ représente $s$, le mot $v$ qui représente $r$ était avant la duplication le plus grand facteur gauche de $\text{suf}(w)$ tel que $|v| = \text{rg}([v])$ d'où $\text{suf}(w) = vb$. La duplication de $d$ transforme donc le lien suffixe sécant de $s$ en un lien suffixe non sécant et, par suite, $s$ n'admet plus de point d'éclatement après la duplication.

(iii): $\forall s \in T \backslash \text{suf}^{-1}(d)$, si $w$ représente $s$, $v$ était le plus grand facteur gauche de suf($w$) tel que $|v| = \text{rg}([v])$ avant la duplication de $d$ et, si $c$ est la lettre de $A$ telle que $vbc$ soit facteur gauche de suf($w$), comme $|vb| = \text{rg}(d')$ et que $\tau(d', c) = \tau(d, c)$ implique $\text{rg}(\tau(d', c)) \geqslant \text{rg}(d) + 1, |vbc| = \text{rg}(d') + 1 < \text{rg}(\tau(d', c))$ après la duplication. $d'$ est alors le point d'éclatement de $s$.

(iv): Si le lien suffixe n'est pas modifié, le point d'éclatement ne l'est pas non plus. Pour tout lien suffixe modifié et distinct de ceux de (iii) et (iv), le lien suffixe était sécant avant la duplication et le reste après et le point d'éclatement reste inchangé. □

**5.8. La procédure PARTAGE.** La procédure PARTAGE a pour fonction de réaliser le partage de $\text{suf}^{-1}(d)$ entre $d$ et $d' = q$ selon le Lemme 5.7. La partie (i) consiste à couper de l'arbre binaire $\mathfrak{B}(d)$ une partie qui deviendra, après insertion de $d$, l'arbre binaire $\mathfrak{B}(d')$.

```
Procedure PARTAGE;
begin h := avs(d); rd := rac(d);
  if rd > 0
  then
  begin u := rd; u1 := alis(u); u2 := u1;
    if avs(u1) >= h
    then
      repeat u := u1; s := u; repeat suf(s) := q; s := δ(s) until s = 0; u1 := alis(u)
      until (avs(u1) < h) or (u = rd);
      if u <> rd then begin alis(rd) := u1; lis(u1) := rd end;
      if avs(u) = h
      then
      begin rac(q) := u; lis(d) := -u; δ(d) := δ(u); δ(u) := d;
        pt(eclat(u)) := q;
        s := u; repeat eclat(s) := 0; s := δ(s) until s = 0;
        if u = rd then rac(d) := 0 else begin alis(u) := u2; lis(u2) := u end;
      end
      else
      begin rac(q) := d; δ(d) := 0; alis(u) := d; alis(d) := u2;
        lis(u2) := d; lis(d) := u; eclat(d) := 0;
      end;
  end;
  else begin rac(q) := d; lis(d) := d; alis(d) := d; δ(d) := 0; eclat(d) := 0 end
end.
```

**5.9. Définition.** Si $(r, b_1 \ldots b_k) \in S \times A^*$ est tel que $\text{rg}(\tau(r, b_1)) > \text{rg}(r) + 1$ et $\tau(r, b_1, \ldots, b_k) \neq \emptyset$, la duplication de $d_1 = \tau(r, b_1)$ selon $(r, b_1)$ suivie par la duplication de $d_2 = \tau(r, b_1 b_2)$ selon $(d_1', b_2)$ où $d_1'$ est le dupliqué de $d$ selon

$(r, b_1), \ldots,$ jusqu'à la duplication de $d_k = \tau(r, b_1 \ldots b_k)$ selon $(d'_{k-1}, b_k)$ où $d'_{k-1}$ est le dupliqué de $d_{k-1}$ selon $(d'_{k-2}, b_{k-1})$ est appelée la *chaîne de duplications selon* $(r, b_1 \ldots b_k)$. Le chemin $(d'_1, d'_2, \ldots, d'_k)$ est appelé le *chemin dupliqué de* $(d_1, d_2, \ldots, d_k)$ *selon* $(r, b_1, \ldots, b_k)$.

Dans les exemples après le Théorème 3.2, le chemin (17, 18, 19) de la Fig. 4 est le chemin dupliqué de (10, 11, 12) selon (6, *abc*).

## 6. Le transfert de mots d'un état à un autre

**6.1. Définition.** Soient $L = \{x_1, \ldots, x_t\}$ avec $x_t$ éventuellement redondant, $a \in A$, $x'_t = x_t a$ et $L' = \{x_1, \ldots, x'_t\}$ sans mots redondants. Deux *états ultimes* distincts $d$ et $d'$ sont dits *quasi-équivalents* si chacun d'eux reconnaît un mot de la suite suffixe ultime $(z_0, z_1, \ldots, z_r)$ de $x_t$. Deux *états non ultimes* distincts $d$ et $d'$ sont dits *quasi-équivalents* s'il existe $a \in A$ tel que les états $\tau(d, a)$ et $\tau(d', a)$ soient quasi-équivalents et, $\forall b \in A \backslash \{a\}$, $\tau(d, b) = \tau(d', b)$.

**6.2. Lemme.** *Les mots les plus courts $u$ et $u'$ reconnus respectivement par les états distincts $d$ et $d'$ dans $\mathfrak{F}(L)$ sont reconnus par un même état dans $\mathfrak{F}(L')$ si, et seulement si, les états $d$ et $d'$ sont quasi-équivalents.*

**Preuve.** (i) En reprenant les notations des Définitions 3.1 et 3.3, $\forall u \in F(L)$, $[u]' \neq [u]''$ si, et seulement si, $\exists v \in [u]' \cap \mathrm{Fg}(z_1, \ldots, z_r)$ tel que $|v| < \mathrm{rg}([v]')$ et, alors $\exists w \in \mathrm{Fd}(x_t)$ et $\exists k, l \in \{1, \ldots, r\}$ avec $k \leqslant l$ tels que, si $u$ est le plus grand mot de $[u]'$, $[u]' = [u]'' \cup (\bigcup_{i=k}^{l} [z_i w^{-1}]'')$ d'après le Théorème 3.4. En outre, $[z_i w^{-1}]'' = [z_i w^{-1}] \cap \mathrm{Fd}(z_i w^{-1})$ d'après le Théorème 3.2.

(ii) Si les mots les plus courts $u$ et $u'$ reconnus respectivement par les états $d$ et $d'$ dans $\mathfrak{F}(L)$ sont reconnus par un même état dans $\mathfrak{F}(L')$, il existe donc $j, j' \in \{1, \ldots, r\}$ et $w \in \mathrm{Fd}(x_t)$ tels que $uw \in L_j \cap \mathrm{Fd}(x_t)$ et $u'w \in L_{j'} \cap \mathrm{Fd}(x_t)$. Si $w = a_1 \ldots a_p$ avec $a_1, \ldots, a_p \in A$, $\forall i \in \{1, \ldots, p\}$, soit $d_i = \tau(d, a_1 \ldots a_i)$ et $d'_i = \tau(d', a_1 \ldots a_i)$. Les états $d_p = [z_j]$ et $d'_p = [z_{j'}]$ sont alors ultimes et quasi-équivalents.

Si $i \in \{1, \ldots, p-1\}$ est tel que les états $d_{i+1}$ et $d'_{i+1}$ soient quasi-équivalents, $\forall b \in A \backslash \{a_{i+1}\}$ tel que $\tau(d_i, b) \neq \emptyset$, $ua_1 \ldots a_i b \in F(L)$ et, comme $[u]' = [u']'$ implique $[ua_1 \ldots a_i b]' = [u'a_1 \ldots a_i b]'$ et que, d'après les Théorèmes 3.2 et 3.4, $[ua_1 \ldots a_i b] = [ua_1 \ldots a_i b]'' = [ua_1 \ldots a_i b]'$ et $[u'a_1 \ldots a_i b] = [u'a_1 \ldots a_i b]'' = [u'a_1 \ldots a_i b]'$, $\tau(d'_i, b) = \tau(d_i, b)$ et les états $d_i$ et $d'_i$ sont aussi quasi-équivalents.

Par récurrence descendante il en résulte que $d$ et $d'$ sont quasi-équivalents.

(iii) Si les états $d$ et $d'$ de $\mathfrak{F}(L)$ sont quasi-équivalents, ou bien $d$ et $d'$ sont ultimes et reconnaissent chacun un mot de $(z_0, \ldots, z_r)$ ou bien il existe des lettres $a_1, \ldots, a_p$ de $A$ telles que, $\forall i \in \{1, \ldots, p\}$, les états $\tau(d, a_1 \ldots a_i)$ et $\tau(d', a_1 \ldots a_i)$ soient quasi-équivalents et qu'en outre les états $\tau(d, a_1 \ldots a_p)$ et $\tau(d', a_1 \ldots a_p)$ soient ultimes d'après la Définition 5.1.

Si $u$ et $u'$ sont respectivement les mots les plus courts reconnus par $d$ et $d'$, $u$ et $u'$ sont reconnus par le même état $[x_t]'$ dans le premier cas. Dans le second cas, $ua_1 \ldots a_p$ et $u'a_1 \ldots a_p$ sont aussi reconnus par l'état $[x_t]'$ et, d'après la minimalité de l'automate $\mathfrak{M}(\tilde{L}')$, il en résulte par récurrence descendante que $u$ et $u'$ sont aussi reconnus par un même état dans $\mathfrak{F}(L')$. □

**6.3. La procédure TESTEFUSION.** La procédure TESTEFUSION est appelée pour un état $t$ et un indice $m$ d'une lettre de $x_t$ tels que les états $\tau(t, x(m))$ et $\text{suf}(\tau(t, x(m)))$ soient quasi-équivalents et sa fonction est de tester si les états $t$ et $\text{suf}(t)$ sont aussi quasi-équivalents.

```
Procedure TESTEFUSION(t, m : integer);
begin d := suf(t);
  if dde(d) = 1
  then fus := 1
  else
    if dde(d) = dde(t)
    then
    begin fus := 1; k := 0;
      while (fus = 1) and (k < carda) do
      begin k := k+1; c := let(k);
        if (τ(d, c) <> τ(t, c)) and (c <> x(m)) then fus := 0;
      end;
    end
    else fus := 0;
end.
```

**6.4. Définition.** Soit $t$ un état de $\mathfrak{F}(L)$ quasi-équivalent à $d = \text{suf(t)}$ et soit $(r, b) \in S \times A$, tel que $d = \tau(r, b)$ et $\text{avs}(t) = \text{pref}(d) - \text{rg}(r) - 1$. La transformation qui retire à l'état $d$ tous les mots $w$ tels que $|w| \leq \text{rg}(r) + 1$ pour les affecter à l'état $t$ est alors appelée le *transfert de mots de d à t selon* $(r, b)$. Lorsque $\text{rg}(r) + 1 < \text{rg}(d)$, le *transfert* est dit *partiel* et, lorsque $\text{rg}(r) + 1 = \text{rg}(d)$, le *transfert* est dit *complet.*

L'automate ainsi transformé reconnaît les mêmes mots que l'automate de départ. Un transfert partiel de $d$ à $t$ selon $(r, b)$ peut être décomposé en la duplication de $d$ selon $(r, b)$ suivie de la fusion de l'état dupliqué de $d$ avec l'état $t$. (Voir la Fig. 8.)

**6.5. La procédure TRANSFERE.** (i) Les fonctions de la procédure TRANSFERE sont proches de celles de la procédure DUPLIQUE sauf pour les points suivants:

- aucun état n'est créé;
- $d = \tau(r0, b)$ est défini à l'aide de $r0$ et non de $r$;
- le transfert explicite d'un mot $u$ de $d$ à $t$ implique $\forall v \in A^*$ tel que $\tau(d, v) \neq \tau(t, v)$ le transfert implicite du mot $uv$ de $\tau(d, v)$ à $\tau(t, v)$;

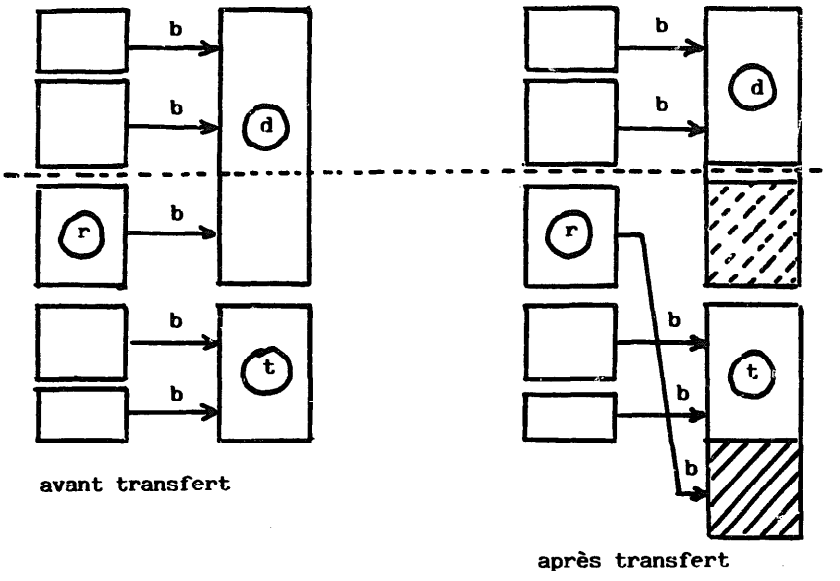


Fig. 8.

- lorsque $\tau(r, b) \neq d$, le transfert se limite à un tel transfert implicite qui découle d'un transfert qui précède alors que, si $\tau(r, b) = d$, tous les mots $w$ reconnus par $d$ et tels que $|w| \geqslant \mathrm{rg}(r)+1$ sont transférés explicitement de $d$ à $t$;
- lorsque $\tau(r, b) = d$, pour tout état $s$ de la suite suffixe de $r$ relative à $b$, $\varphi(s, b)$ subit un décalage de dec = pref($t$) − pref($d$);
- le transfert implique l'échange des liens suffixes entre $d$ et $t$ et cet échange est réalisé dans les arbres $\mathfrak{B}(d)$ et $\mathfrak{B}(t)$ par la procédure ECHANGE qui traite aussi le cas particulier du transfert complet;
- lorsque le transfert est complet, c'est-à-dire lorsque cplet = 1, l'état $d$ est supprimé;
- le transfert d'états de $\mathrm{suf}^{-1}(d)$ à $\mathrm{suf}^{-1}(t)$ est réalisé par la procédure DECPARTAGE qui a une fonction analogue à celle de la procédure PARTAGE.

(ii) Les paramètres de la procédure TRANSFERE sont les états $r0$, $r1$ et $r$ et l'indice $m$ de la lettre $b = x_r(m)$ et sont tels que $d = \tau(r0, b)$ et $t = \tau(r1, b)$ avec $r$ qui pointe sur le mot le plus long qui sera transféré explicitement de $d$ à $t$ lorsque $\tau(r, b) = d$ et $r = r0$ pour un transfert initial (voir Définition 6.10).

```
Procedure TRANSFERE(r, r0, r1, m : integer);
begin b := x(m); d := τ(r0, b); t := τ(r1, b);
  if (cplet = 1) and (ψ(r0, b) > 0)
  then begin cplet := 0; dpred(d) := ψ(r0, b); ψ(r0, b) := 0 end
  ECHANGE(t, d); suf(t) := suf(d); h := avs(t); avs(t) := avs(d); avs(d) := h;
  if cplet = 0 then begin suf(d) := t; if r = r0 then dpred(d) := ψ(r, b) end;
  eclat(t) := eclat(d); if τ(r, b) = d then ψ(r, b) := r1;
  dec := pref(t) − pref(d); DECPARTAGE;
  for k := 1 to carda do
```

```
begin c := let(k); s := τ(d, c);
  if (s>0) and (c<>x(m+1))
  then
  begin if ψ(suf(t), c) = d then ψ(suf(t), c) := t
    if dpred(s) = d then dpred(s) := t;
  end;
end;
s := r; while τ(s, b) = d do
  begin τ(s, b) := t; φ(s, b) := φ(s, b) + dec; dpred(t) := s; s := suf(s) end;
r2 := τ(s, b);
end.
```

**6.6. Lemme.** *Lors d'un transfert de mots de d à t selon* $(r, b)$:

(i) *l'état t hérite de l'éventuel point d'éclatement de d et, lorsque le transfert est partiel, d hérite de celui de t*;

(ii) *lorsque le transfert est complet un nouveau point d'éclatement est affecté à l'ensemble T des descendants des états* $s \neq t$ *de* $\text{suf}^{-1}(d)$ *dont le lien suffixe n'était pas sécant*;

(iii) *aucun autre point d'éclatement n'est créé ou modifié.*

**Preuve.** (i): Si $u$, $u'$ et $v$ sont les mots qui représentent respectivement les états $d$, $t$ et $r$; si, avant le transfert, $d$ admettait un point d'éclatement rd, alors $|\text{suf}(u)| < \text{rg}([\text{suf}(u)])$ et comme la valeur de $\text{suf}(u)$ est transférée à $\text{suf}(u')$, $t$ admettra, après le transfert, rd comme point d'éclatement.

Si le transfert est partiel comme, avant le transfert, $\text{suf}(u') = vb$ avec $|vb| < \text{rg}(d)$, $t$ admettait $r$ comme point d'éclatement et, après le transfert du mot $vb$ à $t$, $|vb| < |u'| = \text{rg}(t)$ et $\text{suf}(u) = vb$. $r$ sera donc le point d'éclatement de $d$.

(ii): Si le transfert est complet et si, avant le transfert, $T_0 = \Delta(t) \backslash \{t\} \neq \emptyset$, $\forall t_0 \in T_0$, si $u_0$ représente $t_0$, alors, $\text{suf}(u_0) = u = vb$ et, après le transfert, $u$ a été transféré à $t$ et $|u| < |u'|$: $t_0$ qui n'avait pas de point d'éclatement avant en admettra donc un après. D'après le Lemme 5.2, il en est de même de tous les descendants de $t_0$.

(iii): Aucun autre lien suffixe non sécant n'est créé et tous les autres liens suffixes sécants le restent. Aucun autre point d'éclatement n'est donc créé ou modifié. □

**6.7. La procédure DECPARTAGE.** (i) La fonction principale de cette procédure est de soustraire à $\text{suf}^{-1}(d)$ des états pour les adjoindre à $\text{suf}^{-1}(t)$ selon le Lemme 6.6 et elle le réalise en enlevant à $\mathfrak{B}(d)$ un sous-arbre et en le greffant sur $\mathfrak{B}(t)$. Elle réalise simultanément un décalage de $\text{dec} = \text{pref}(t) - \text{pref}(d)$ sur $\text{avs}(s)$ pour tout état $s$ qui passe de $\text{suf}^{-1}(d)$ à $\text{suf}^{-1}(t)$ et réalise la création de nouveaux points d'éclatement ou leur redéfinition selon le Lemme 6.6.

(ii) La procédure DECPARTAGE est interne à la procédure TRANSFERE. La valeur de $e0$ est fournie par la procédure DUPLIFERE.

```
Procedure DECPARTAGE;
begin rd := rac(d); u := rd; v1 := alis(rd); rt := rac(t); h := h + dec;
  repeat u := alis(u); s := u;
    repeat suf(s) := t; avs(s) := avs(s) + dec; s := δ(s) until s = 0;
  until avs(u) = h;
  if u = rd
  then
  begin rac(d) := 0; if cplet = 1 then
    begin s := u; repeat eclat(s) := e0; s := δ(s) until s = 0 end;
   end
   else begin u1 := alis(u); alis(rd) := u1; lis(u1) := rd end;
   if rt > 0 then
   begin v2 := alis(rt); lis(v2) := u; alis(u) := v2;
     lis(v1) := rt; alis(rt) := v1
   end
   else begin rac(t) := u; alis(u) := v1; lis(v1) := u end;
end.
```

**6.8. Définition.** Si les états $f_0$ et $\mathrm{suf}(f_0)$ sont quasi-équivalents, soit $(f_0, f_1, \ldots, f_k)$ la suite d'états telle que, $\forall i \in \{0, \ldots, k-1\}$, $f_{i+1} = \mathrm{dpred}(f_i)$ et les états $f_i$ et $\mathrm{suf}(f_i)$ sont quasi-équivalents alors que $f_k$ et $\mathrm{suf}(f_k)$ ne sont pas quasi-équivalents. L'état $f_k$ est alors appelé le *point de fusion de* $f_0$.

**6.9. Lemme.** *Si* $v$ *est le plus grand facteur gauche de* $\mathrm{suf}(x_t)$ *tel que* $|v| = \mathrm{rg}([v])$ *et si* $\mathrm{re} = [v]$ *alors*

$$\mathrm{pref}_t(v) + \mathrm{pref}(\mathrm{suf}(f_0)) = n - 1 + \mathrm{avs}(f_0) + \mathrm{rg}(\mathrm{re}).$$

**Preuve.** Si $u$ est le mot qui représente $f_0$, le point d'éclatement re de $f_0$ reconnaît le plus grand facteur gauche $v$ de $\mathrm{suf}(u)$ tel que $|v| = \mathrm{rg}([v]) = \mathrm{rg}(\mathrm{re})$ d'après la Définition 5.1. Si $\mathrm{pref}_t(v) = a_1 \ldots a_{\mathrm{ne}-1}$, $v^{-1}\mathrm{suf}(u) = a_{\mathrm{ne}} \ldots a_{n-1}$ d'où $n - \mathrm{ne} = |\mathrm{suf}(u)| - |v|$ et, comme $|\mathrm{suf}(u)| = \mathrm{pref}(\mathrm{suf}(f_0)) - \mathrm{avs}(f_0)$, $\mathrm{ne} = n - \mathrm{pref}(\mathrm{suf}(\mathrm{f}_0)) + \mathrm{avs}(f_0) + \mathrm{rg}(\mathrm{re})$. □

**6.10. La chaîne de duplications transferts et la procédure DUPLIFERE.** (i) Soient $x_t = a_1 \ldots a_{n-1}$ et $x'_t = x_t a_n$ avec $a_1, \ldots, a_n \in A$ et les états ultimes $f_0 = [x_t]$ de $\mathfrak{F}(L)$ et $f = [x'_t]'$ de $\mathfrak{F}(L')$. La procédure DUPLIFERE est appelée lorsque $\mathrm{suf}(f_0)$ est un état ultime de $\mathfrak{F}(L)$, c'est-à-dire lorsque $f_0$ et $\mathrm{suf}(f_0)$ sont quasi-équivalents et sa première fonction est de déterminer le point de fusion rf de $f_0$ et l'indice nf à l'aide de la procédure TESTEFUSION ainsi que le point d'éclatement re de $f_0$ et l'indice $\mathrm{ne} = \mathrm{pref}_t(v) + 1$ selon le Lemme 6.9.

(ii) La fonction principale de cette procédure est de réaliser une chaîne de duplications-transferts. Lorsque $\mathrm{ne} < \mathrm{nf}$, elle réalise d'abord une chaîne de duplica-

tions selon $(re, a_{ne} \dots a_{nf-1})$, puis une chaîne de transferts. Lorsque $nf \leq ne$ elle se limite à une chaîne de transferts et lorsque $nf < ne$ elle commence par $ne - nf$ transferts complets. Dans tous les cas, suf(rf) est le nouveau point d'éclatement de $f_0$.

**Procedure** DUPLIFERE;
**begin** rf := $f0$, nf := $n$; fus := 1;
  **while** fus = 1 **do begin** rf := dpred(rf): nf := nf − 1; TESTEFUSION(rf, nf) **end**;
  $e0$ := eclat($f0$); re := pt($e0$); ne := $n$ − pref(suf($f0$)) + avs($f0$) + rg(re);
  **if** nf < ne **then begin** pt($e0$) := suf(rf); cplet := 1 **end else** cplet := 0;
  **while** ne < nf **do begin** DUPLIQUE(re, ne); re := $q$; ne := ne + 1 **end**;
  $r$ := suf(rf); $r1$ := rf; $r0$ := $r$;
  **while** nf < $n$ **do**
    **begin** TRANSFERE($r$, $r0$, $r1$, nf); $r$ := $r2$; $r0$ := $d$; $r1$ := $t$; nf := nf + 1 **end**;
**end**.

**Exemple.** Si $x_1 = abbcabca$, $x_2 = abbabca$, $x_3 = abbabcb$, $x_4 = ababca$ et $L = \{x_1, x_2, x_3, x_4\}$, le graphe de $\mathfrak{F}(L)$ est donné par la Fig. 9. Si $x_4' = x_4 b$ et si $L' = \{x_1, x_2, x_3, x_4'\}$, le graphe de $\mathfrak{F}(L')$ est donné par la Fig. 10.

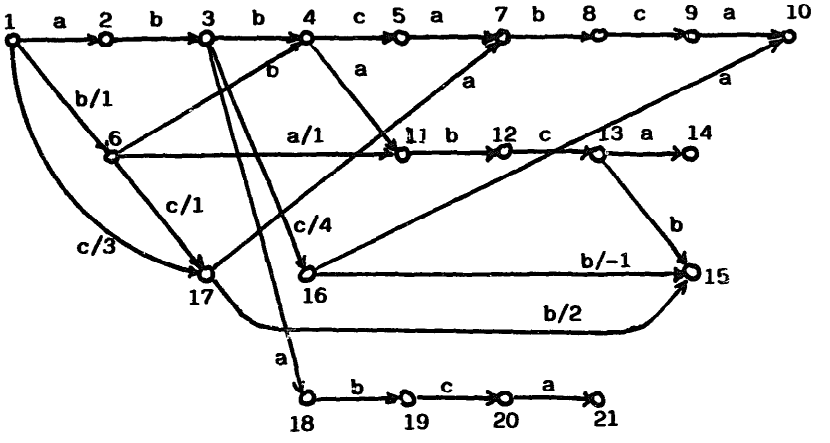


Fig. 9.

La procédure DUPLIFERE est appelée pour $f0 = 21$ car $14 = \text{suf}(21)$ est un état ultime. Alors re = 6 et rf = 20 et ne = 3 < nf = 6 et elle réalise donc la duplication de (11, 12, 13) en (23, 24, 25) selon (6, *abc*) puis le transfert du mot *babca* de l'état 14 à l'état 21.

La procédure DUPLIFERE est alors appelée une seconde fois car alors suf(21) = 10 est encore un état ultime (voir procédure PROLONGE en Définition 7.3). Alors re = 16 et rf = 24 et nf = 4 < ne = 5 et elle réalise donc le transfert complet de l'état 16 à l'état 25 puis le transfert du mot *abca* de l'état 10 à l'état 21.

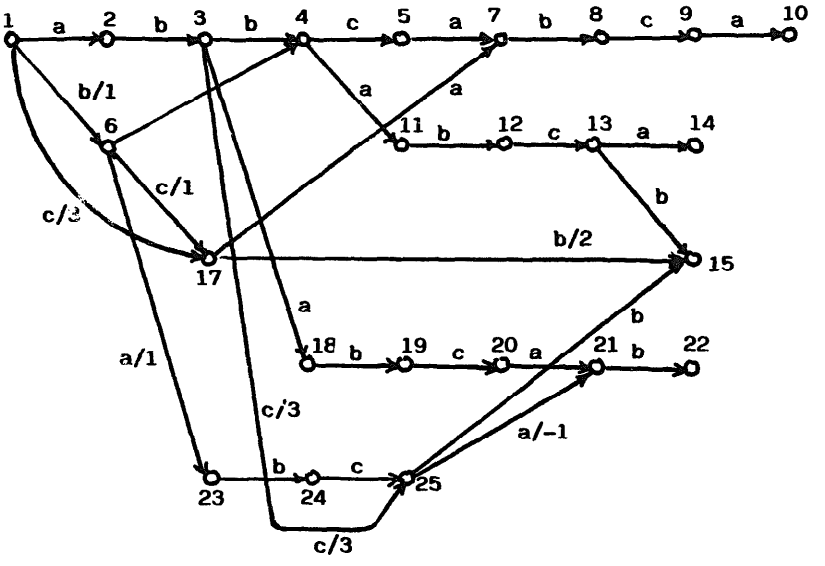


Fig. 10.

## 7. La construction en ligne du transducteur de progression $\mathfrak{F}(L)$

**7.1. Les duplications frontales.** Soit $L=\{x_1,\dots,x_t\}$ avec $|x_1|\geq\cdots\geq|x_{t-1}|>|x_t|$ et $x_t$ redondant pour $L$ et soit $a\in A$ tel que $x'_t=x_ta$ ne soit pas redondant pour $L'=\{x_1,\dots,x_{t-1},x'_t\}$.

Dans le cas où $|x_t|<\mathrm{rg}([x_t])$, l'état $[x_t]$ doit être dupliqué car $x'_t\in F(L')$ alors que, pour tout mot $u$ de $[x_t]$ tel que $|x_t|<|u|$, $ua\notin F(L')$. D'après le Théorème 3.2, si $x_t=\dots a_{n-1}$ avec $a_1,\dots,a_{n-1}\in A$, si $n0$ est le plus grand entier de $\{1,\dots,n-1\}$ tel que $n0=\mathrm{rg}([a_1\dots a_{n0}])$ et si, $\forall i\in\{n0,\dots,n-1\}$, $v_i=a_1\dots a_i$, $[v_i]'=[v_i]\cap \mathrm{Fd}(v_i)$, alors l'état $[v_{n0+1}]$ doit être dupliqué selon $([v_{n0}],a_{n0+1})$, l'état $[v_{n0+2}]$ doit être dupliqué selon $([v_{n0+1}]',a_{n0+2})$ et ainsi de suite jusqu'à l'état $[v_{n-1}]$ qui doit être dupliqué selon $([v_{n-2}]',a_{n-1})$. Les *duplications* ainsi définies sont dites *frontales* et la succession de ces duplications est appelée la *chaîne de duplications frontales selon* $([v_{n0}],a_{n0+1}\dots a_{n-1})$.

Par exemple, si $x=abbabc$ et $y=bababc$ et si $L=\{x,y\}$ comme dans l'exemple après la Definition 1.2 (Fig. 1) le facteur gauche *baba* de $y$ provoque la chaîne de duplications frontales selon $(6,ab)$ formée de la duplication de 5 selon $(6,a)$ suivie de la duplication de 7 selon $(9,b)$.

**7.2. La création d'un nouvel état ultime.** La procedure ULTIME est appelée lorsque le transducteur en mémoire est $\mathfrak{F}(L)$ et que $f0$ est l'état ultime qui reconnaît le dernier mot $x_t$ de $L$ et lorsque ce transducteur se déduit de $\mathfrak{F}(L)$ par la chaîne de duplications frontales selon $([v_{n0}],a_{n0+1}\dots a_{n-1})$ avec les notations de la Définition 7.1 et $f0=[v_{n-1}]'$. Sa fonction est de créer un nouvel état ultime $f$ qui reconnaît le mot $x'_t$ et le langage $L'_t=\mathrm{Fd}(x'_t)\backslash F(L)$ à partir de ce transducteur.

Lorsque $\tau(\text{suf}(f0), a) \neq \emptyset$, la création de $f$ de la transition $\tau(f0, a) = f$ implique que $\text{suf}(f) = \tau(\text{suf}(f0), a)$ et que le langage reconnu par $f$ est $L'_t = \text{Fd}(x'_t) \setminus F(L) = [x_t]'a$ et le transducteur ainsi construit est isomorphe à $\mathfrak{F}(L')$. Le cas où $\tau(\text{suf}(f0), a) = \emptyset$ est étudié en Définition 7.3.

**Procedure** ULTIME;
**begin** $q := q+1$; $\tau(f0, a) = q$; $\text{dde}(f0) := \text{dde}(f0)+1$; $\text{dde}(q) := 0$; $\text{rac}(q) := 0$;
  **for** $k := 1$ **to** carda **do begin** $\tau(q, \text{let}(k)) := 0$; $\psi(q, \text{let}(k)) := 0$ **end**;
  $\text{rg}(q) := n$; $\text{pref}(q) := n$; $\text{part}(q) := \text{nbm}$; $\varphi(f0, a) := n - \text{pref}(f0) - 1$; $f := q$;
  **if** $\tau(\text{suf}(f0), a) > 0$
  **then**
  **begin** $\text{suf}(f) := \tau(\text{suf}(f0), a)$; $\text{dpred}(f) := f0$;
    $\text{avs}(f) := \text{avs}(f0) + \varphi(\text{suf}(f0), a)$; INSERE($f$);
    **if** $\text{lis}(f) > 0$
    **then** POINTDECLAT($f0, f, a$) **else** $\text{eclat}(f) := \text{eclat}(-\text{lis}(f))$;
  **end**
  **else** PROLONGE;
  $f0 := f$;
**end.**

**7.3. La prolongement ultime.** Dans le cas où $\tau(\text{suf}(f0), a) = \emptyset$, la suite suffixe $(z_0, \ldots, z_s)$ de $x_t$ relative à la lettre $a$ n'est pas réduite à $z_0 = x_t$, et $L'_t = \text{Fd}(x'_t) \setminus F(L) = \bigcup_{i=0}^{s} ([z_i] \cap \text{Fd}(x_t)).a$ comme en Définition 3.1. $([z_0), \ldots, [z_s])$ est alors la suite suffixe de $f0 = [z_0]$ relative à $a$.

D'après les Théorèmes 3.2 et 3.4, si $(z_0, \ldots, z_r)$ est la suite suffixe ultime de $x_t$, c'est-à-dire si, $\forall i \in \{0, \ldots, r\}$, $[z_i]$ est un état ultime de $\mathfrak{F}(L)$ et, lorsque $s > r$, l'état $[z_{r+1}]$ n'est pas ultime, alors $\forall i \in \{1, \ldots, r\}$, une chaîne de duplications-transferts se termine par le transfert de tous les mots de $[z_i] \cap \text{Fd}(x_t)$ de l'état $[z_i]$ à l'état $f0$ et, en posant $\tau(f0, a) = f$, $f$ reconnaît tous les mots de $([z_i] \cap \text{Fd}(x_t)).a$.

En outre, $\forall i \in \{r+1, \ldots, s\}$ tel que $[z_{i-1}]$ possède un point d'éclatement, une chaîne de duplications issue de ce point se termine par la duplication de $[z_i]$ en deux états dont le second $[z_i]'$ reconnaît le langage $[z_i] \cap \text{Fd}(x_t)$ et, en posant $\tau([z_i]', a) = f$, $f$ reconnaît encore tous les mots de $([z_i] \cap \text{Fd}(x_t)) \,.\, a$.

Enfin, $\forall i \in \{r+1, \ldots, s\}$ tel que $[z_{i-1}]$ ne possède pas de point d'éclatement, $z_i$ est le plus grand mot de $[z_i]$ et, en posant $\tau([z_i], a) = f$, $f$ reconnaît aussi les mots de $([z_i] \cap \text{Fd}(x_t)).a$.

L'ensemble de ces transformations est appelé le *prolongement ultime de f0 relatif à la lettre a* et la fonction de la procédure PROLONGE est de réaliser ce prolongement ultime. Il résulte des Théorèmes 3.2 et 3.4 que le transducteur ainsi construit est bien $\mathfrak{F}(L')$ (à un isomorphisme près).

**Procedure** PROLONGE;
**begin while** $\text{dde}(\text{suf}(f0)) = 0$ **do** DUPLIFERE; $s := f0$;
  **while** $\tau(\text{suf}(s), a) = 0$ **do**

```
begin r := pt(eclat(s));
  if r > 0 then
  begin m := n − pref(suf(s)) + avs(s) + rg(r);
    while m < n do begin DUPLIQUE(r, m); r := q; m := m + 1 end;
  end;
  ψ(suf(s), a) := s; s := suf(s); τ(s, a) := f; dde(s) := dde(s) + 1;
  φ(s, a) := pref(f) − pref(s) − 1;
end;
avs(f) := avs(s) + φ(suf(s), a); suf(f) := τ(suf(s), a); dpred(f) := s; INSERE(f);
if lis(f) > 0 then POINTDECLAT(s, f, a) else eclat(f) := eclat(−lis(f));
end.
```

**Exemple.** Si $x_1 = abbcabc$, $x_2 = abbabc$ et $x_3 = ababc$, $L = \{x_1, x_2, x_3\}$, $x_3' = x_3 b$ et $L' = \{x_1, x_2, x_3'\}$ comme dans les Théorèmes 3.2 et 3.4 (voir Figs. 3 et 5) la procédure PROLONGE est appelée pour $f0 = 15$ et la lettre $b$.

Comme $\text{suf}(f0) = 12$ et $\text{dde}(12) = 0$, la procédure DUPLIFERE engendre le transfert du mot $ba$ de l'état 10 à l'état 13 lors du premier appel et alors $\text{suf}(f0) = 9$ avec $\text{dde}(9) = 0$ et le deuxième appel de la procédure DUPLIFERE provoque le transfert du mot $abc$ de l'état 9 à l'état 15 et alors $\text{suf}(f0) = 5$ avec $\text{dde}(5) = 1$ et le point d'éclatement de $f0 = 15$ est alors 6. La duplication de l'état 5 selon $(6, c)$ conduit alors à la création de l'état 17 et la transition créée $\tau(17, b) = 16$ termine les calculs.

**7.4. La construction en ligne de $\mathfrak{F}(L)$.** La procédure TRANSPROGRESSE réalise la construction en ligne du transducteur de progression d'un langage fini en éliminant les mots redondants lorsque les mots sont rangés selon leurs longueurs décroissantes au sens large: $|x_1| \geqslant x_2| \geqslant \cdots \geqslant |x_t|$.

Lorsque $x_t' = x_t a$ est substitué à $x_t$, les cas suivants peuvent se produire.

(i) Le mot $x_t'$ est redondant pour $L'$, c'est-à-dire reconnu par $\mathfrak{F}(L)$ et ceci est testé par $\text{fprog} = 0$. Dans ce cas, $\mathfrak{F}(L)$ reste inchangé et l'élimination des mots redondants en découle. En outre, si $x_t' = a_1 \ldots a_n$ avec $a_1, \ldots, a_n = a \in A$, le plus grand entier $n0$ tel que $n0 = \text{rg}([a_1 \ldots a_{n0}])$ et l'état $f0 = [a_1 \ldots a_{n0}]$ sont déterminés afin de préparer les éventuelles duplications frontales à réaliser dans le cas suivant.

(ii) Le mot $x_t$ est redondant pour $L$ mais $x_t'$ ne l'est pas pour $L'$ et ce cas correspond à $\text{fprog} = 1$. Dans ce cas ou bien $n0 = n - 1$ et $f0$ reconnaît le mot $x_t$ ou bien $n0 < n - 1$ et la chaîne de duplications frontales selon $(f0, a_{n0+1} \ldots a_{n-1})$ est réalisée et, après avoir redéfini $f0$ comme l'état qui reconnaît le mot $x_t$, un nouvel état ultime est créé et le prolongement ultime associé est réalisé.

(iii) Le mot $x_t$ n'est pas redondant pour $L$ et alors $x_t' = x_t a$ ne l'est pas non plus pour $L'$ et ce cas correspond donc à $\text{fprog} > 1$. Dans ce cas, on est ramené au cas précédent mais sans les duplications frontales.

D'après les Théorèmes 3.2 et 3.4 et les justifications associés aux différentes procédures, la procédure TRANSPROGRESSE calcule bien le transducteur de progression d'un langage fini quelconque.

```
Procedure TRANSPROGRESSE;
begin readln(nbmot); alpha:=[ ]; carda:=0; q:=1; e:=0;
  rg(0):=-1; pref(0):=-1; suf(0):=0; dde(0):=0; rg(1):=0; pref(1):=0;
  suf(1):=0; avs(1):=0; part(1):=1; rac(1):=0;
  for nbm:=1 to nbmot do
  begin f:=1; f0:=1; n0:=0; fprog:=0; readln(lgx);
    for n:=1 to lgx do
    begin LIRE; τ(f0, a):=0 then fprog:=fprog+1;
      if fprog=0 then
      begin
        if (n0=n-1) and (rg(τ(f, a))=rg(f)+1)
        then begin n0:=n; f0:=τ(f, a) end;
        f:=τ(f, a);
      end
      else
      begin if fprog=1 then while n0<n-1 do
        begin n0:=n0+1; DUPLIQUE(f0, n0); f0:=q end;
        ULTIME;
      end;
    end;
  end;
end.
```

La procédure LIRE est interne à la procédure TRANSPROGRESSE et sa fonction essentielle est de lire une lettre d'un mot de $L$ mais elle réalise aussi l'initialisation de $\tau$ et met les lettres de $A$ dans un ensemble alpha et dans un tableau let.

```
Procedure LIRE;
begin read(a); x(n):=a;
  if not (a in alpha) then
  begin τ(0, a)=1; φ(0, a)=0; dde(0):=dde(0)+1;
    for s:=1 to carda do τ(s, a):=0;
    alpha:=alpha+[a]; carda:=carda+1; let(carda):=a;
  end;
end.
```

## 8. La complexité de la construction de $\mathfrak{F}(L)$

**8.1. Lemme.** *Si $\rho(L, L')$ est le nombre de transitions créées ou redéfinies lors de la transformation de $\mathfrak{F}(L)$ en $\mathfrak{F}(L')$ et si $\delta_1(L, L')$ et $\delta_2(L, L')$ sont respectivement le nombre de duplications et le nombre de transferts réalisés lors de cette transformation alors*

$$\rho(L, L') \leqslant (1+|A|)\delta_1(L, L') + \delta_2(L, L') + |L'_t| - |L_t| + 1.$$

**Preuve.** (i) Lorsque $d'$ est le dupliqué d'un état $d$, $\forall c \in A$ tel que $\tau(d, c) \neq \emptyset$, la transition $\tau(d, c)$ est aussi dupliquée en $\tau(d', c) = \tau(d, c)$ et, $\forall c \in A$ tel que $\tau(d, c) = \emptyset$, $\tau(d', c)$ est initialisé à $\emptyset$. Il existe $\delta_1(L, L') \cdot |A|$ transitions ainsi définies ou initialisées.

(ii) Lorsque $d'$ est le dupliqué d'un état $d$ selon $(r, b)$ ou lorsqu'il y a transfert de tous les mots y reconnus par $d$ et tels que $|y| \leqslant \mathrm{rg}(r)+1$ à un état $d'$, la transition $\tau(r, b) = d$ est redéfinie en $\tau(r, b) = d'$ et il existe $\delta_1(L, L') + \delta_2(L, L')$ transitions de ce type.

(iii) Si $(z_0, \ldots, z_s)$ est la suite suffixe de $x_t$ relative à la lettre $a$ dans $\mathfrak{F}(L)$ et si $u$ est le représentant d'un état $d$ qui est dupliqué selon $(r, b)$ ou qui perd dans un transfert tous les mots $y$ tels que $|y| \leqslant \mathrm{rg}(r)+1$, il existe $i \in \{0, \ldots, s\}$ et $v \in [u] \cap \mathrm{Fg}(z_i)$ tel que $|v| < \mathrm{rg}(d)$ et alors, si $w = v^{-1} z_i$, $([u] \cap \mathrm{Fd}(v)).wa \subseteq L'_t$ d'après la démonstration du Théorème 3.2. De plus, si $i = 0$, un tel mot $v$ n'existe que si le mot $x_t$ est redondant, c'est-à-dire si $L_t = \emptyset$.

Pour tout état $s \neq r$ de la suite suffixe de $r$ relative à la lettre $b$, la transition $\tau(s, b) = d$ est redéfinie en $\tau(s, b) = d'$. Si $u_s$ est le mot qui représente l'état $s$, le mot $u_s bw$ est reconnu par l'état $[z_i]$ de $\mathfrak{F}(L)$ et $u_s bwa \in L'_t$ d'après la Définition 3.1. En outre, $u_s bwa \notin L_t a$ puisque $L_t = \emptyset$ lorsque $i = 0$. En outre, $\forall i \in \{1, \ldots, s\}$, une transition est créée de l'état $[z_i]'$ vers l'état ultime $f = [x'_t]'$ de $\mathfrak{F}(L')$ par la procédure PROLONGE et cette transition est associée au mot $z_i a$ de $L'_t \backslash L_t a$.

Comme tous les mots de $L'_t \backslash L_t a$ ainsi obtenus sont deux à deux distincts, le nombre de ces transitions redéfinies et de ces transitions créées est majoré par $|L'_t \backslash L_t a| + 1 = |L'_t| - |L_t| + 1$ en comptant la transition créée $\tau(f0, a) = f$.

(iv) Le cumul des résultats de (i), (ii) et (iii) donne alors

$$\rho(L, L') \leqslant (1+A)\delta_1(L, L') + \delta_2(L, L') + |L'_t| - |L_t| + 1. \qquad \square$$

**8.2. Lemme.** *Le nombre de duplications et de transferts réalisés lors de la construction de* $\mathfrak{F}(L)$ *est majoré par* $4\|L\| - 7|L|$.

**Preuve.** Soient $\eta_1$ le nombre d'états ultimes créés, $\eta_2$ le nombre de duplications, $\eta_3$ le nombre de transferts partiels et $\eta_4$ le nombre de transferts complets qui sont intervenus lors de la construction de $\mathfrak{F}(L)$.

(i) Le nombre d'états de $\mathfrak{F}(L)$ est alors $\eta_1 + \eta_2 - \eta_4 + 1$ et $\eta_1 + \eta_2 - \eta_4 \geqslant |L| + \mathrm{lgmax} - 1 \geqslant |L|$ d'après le Théorème 2.3. Le nombre total de duplications et de transferts vérifie donc

$$\eta = \eta_2 + \eta_3 + \eta_4 \leqslant \eta_1 + 2\eta_2 + \eta_3 - |L|.$$

(ii) Comme $\mathfrak{F}(L)$ se déduit de l'automate $\mathfrak{M}(\tilde{L})$ en supprimant l'état qui reconaît $A^* \backslash F(L)$ et que $\mathfrak{M}(\tilde{L})$ est un quotient de l'automate $\mathfrak{M}(\tilde{J})$, où $\tilde{J} = (F(x_1), \ldots, F(x_t))$ d'après la Proposition 1.3, $\forall u \in F(L)$, les classes $[u]_J$ et $[u]$ de $u$ modulo $\mathrm{Pd}(\tilde{J})$ et $\mathrm{Pd}(\tilde{L})$ vérifient $[u]_J \subseteq [u]$.

(iii) Lors du transfert de mots d'un état $d$ à un état $d'$ selon $(r, b)$, si $u$, $u'$ et $v$ sont les représentants respectifs de $d$, $d'$ et $r$, tous les mots de $[u] \cap \mathrm{Fd}(vb)$ sont

transférés de $d$ à $d'$. Il existe alors $i \in \{1, \ldots, t-1\}$ tel que $vb \in F(x_i)$ et $u' \notin F(x_i)$ et, comme $\mathrm{Pd}(\tilde{J}) = \bigcup_{i \in 1, \ldots, t} \mathrm{Pd}(F(x_i))$, $[u']_J \neq [vb]_J$, alors que $[u'] = [vb]$.

Si plusieurs classes de mots sont successivement transférés au même état $d'$, les indices $i$ associés sont deux à deux distincts d'après la démonstration du Théorème 3.4 et les classes modulo $\mathrm{Pd}(\tilde{J})$ le sont donc aussi.

(iv) Il résulte de (ii) et (iii) que $\mathfrak{M}(\tilde{J})$ contient au moins $\eta_1 + \eta_2 + \eta_3 + 1$ états distincts d'où $\eta_1 + \eta_2 + \eta_3 \leqslant 2\|L\| - 3|L| + 1$ d'après le Lemme 2.1. Il en résulte, d'après (i), que $\eta \leqslant 4\|L\| - 7|L|$. □

**8.3. Lemme.** *$s$ étant un état de $\mathfrak{F}(L)$, si le lien suffixe de $s$ est sécant alors $|\Delta(s)| < |A|$ et sinon, $|\Delta(s)| \leqslant |A|$.*

**Preuve.** (i) Si $u_1$ et $u_2$ sont les mots les plus courts reconnus respectivement par les états $t_1$ et $t_2$ de $\Delta(s)$,

$$|\mathrm{suf}(u_1)| = \mathrm{pref}(\mathrm{suf}(s)) - \mathrm{avs}(t_1) = \mathrm{pref}(\mathrm{suf}(s)) - \mathrm{avs}(t_2) = |\mathrm{suf}(u_2)|$$

et, comme $\mathrm{suf}(t_1) = \mathrm{suf}(t_2)$, $\mathrm{suf}(u_1) = \mathrm{suf}(u_2)$ d'après la Proposition 1.4 et $\exists a_1, a_2 \in A$ tels que, en posant $u = \mathrm{suf}(u_1)$, $u_1 = a_1 u$ et $u_2 = a_2 u$. En outre, $t_1 \neq t_2$ implique $a_1 \neq a_2$ et, par suite, $|\Delta(s)| \leqslant |A|$.

(ii) Lorsque le lien suffixe de $s$ est sécant, $u$ n'est pas le mot le plus long reconnu par $\mathrm{suf}(s)$ et il existe $a \in A$ tel que le mot $au$ soit aussi reconnu par $\mathrm{suf}(s)$. Comme $\mathrm{suf}(s) \notin \Delta(s)$, $a_1 \neq a$ et $a_2 \neq a$ et, par suite, $|\Delta(s)| < |A|$ dans ce cas. □

**8.4. Définition.** (i) Une suite $(s_1, \ldots, s_k)$ d'états de $\mathfrak{F}(L)$ est un chemin de $\mathfrak{F}(L)$ si, et seulement si, il existe des lettres $a_2, \ldots, a_k$ telles que, $\forall i \in \{1, \ldots, k-1\}$, $\tau(s_i, a_{i+1}) = s_{i+1}$. Lorsque le dernier état $s_k$ est l'état ultime qui reconnaît le mot $x$ de $L$, le *chemin* $(s_1, \ldots, s_k)$ est dit *ultime* et *associé à $x$.*

(ii) Un *chemin* $(s_1, \ldots, s_k)$ est dit *sécant* si, $\forall i \in \{1, \ldots, k\}$, le lien suffixe de $s_i$ est sécant.

**8.5. Lemme.** *Tout état $s$ dont le lien suffixe est sécant appartient à un chemin sécant associé à un mot de $L$ et, pour chaque mot $x$ de $L$, il existe au plus un chemin sécant maximal associé à $x$.*

**Preuve.** (i) Si le lien suffixe de $s$ est sécant, $s$ et tous ses descendants dans $\mathfrak{F}(L)$ admettent un même point d'éclatement d'après le Lemme 5.2. Si $u$ est le représentant de $s$ et si $w = a_1 \ldots a_k$ où $a_1, \ldots, a_k \in A$ est un mot de longueur maximale tel que $uw \in F(L)$, $[uw]$ est un état ultime et il existe $x \in L$ tel que $[uw] = [x]$. Si, $\forall i \in \{1, \ldots, k\}$, $s_i = [ua_1 \ldots a_i]$, alors $(s, s_1, \ldots, s_k)$ est un chemin sécant ultime associé à $x$.

(ii) Soient $s_1$ et $s_2$ deux états dont les liens suffixes sont sécants et tels qu'il existe $a \in A$ tel que $\tau(s_1, a) = \tau(s_2, a)$. Si $u_1$ et $u_2$ sont les mots les plus courts reconnus respectivement par $s_1$ et $s_2$, $\exists a_1, a_2 \in A$ tels que $u_1 = a_1 \mathrm{suf}(u_1)$ et $u_2 = a_2 \mathrm{suf}(u_2)$.

D'après (i), $s_1$ et $s_2$ admettent chacun un point d'éclatement et, d'après le Lemme 5.2,

$$\tau(\mathrm{suf}(s_1), a) = \mathrm{suf}(\tau(s_1, a)) = \mathrm{suf}(\tau(s_2, a)) = \tau(\mathrm{suf}(s_1), a).$$

$t$ reconnaît alors les mots $u_1a$ et $u_2a$ alors que $\mathrm{suf}(t)$ reconnaît les mots $\mathrm{suf}(u_1)a$ et $\mathrm{suf}(u_2)a$. Il en résulte que $|u_1a| = |u_2a|$ ce qui implique $u_1 = u_2$ d'après la Proposition 1.4 et $s_1 = s_2$.

Tout état dont le lien suffixe est sécant admet donc au plus un prédécesseur qui a la même propriété et ceci implique que, pour chaque mot $x$ de $L$, il existe au plus un chemin sécant maximal associé à $x$. □

**8.6. Lemme.** *$\mathfrak{F}(L)$ contient au plus $\|L\| - 2|L|$ liens suffixes sécants.*

**Preuve.** D'après le Lemme 8.5, pour tout mot $x$ de $L$, il existe au plus un chemin sécant maximal associé à $x$. Comme, $\forall a, b \in A$, le lien suffixe de $[ab]$ est non sécant, la longueur de ce chemin est majorée par $|x| - 2$. Il en résulte que $\mathfrak{F}(L)$ contient au plus $\|L\| - 2|L|$ liens suffixes sécants. □

**8.7. Lemme.** *Pour tout état $d$ de $\mathfrak{F}(L)$, $\mathrm{suf}^{-1}(d)$ contient au plus $|L|$ liens suffixes sécants.*

**Preuve.** D'après le Lemme 8.5, il existe, pour tout mot $x$ de $L$, au plus un chemin sécant maximal $(s_1, \ldots, s_k)$ associé à $x$. Si $a_2, \ldots, a_k$ sont les lettres de $A$ telles que, $\forall i \in \{1, \ldots, k-1\}$, $\tau(s_i, a_{i+1}) = s_{i+1}$, alors $\tau(\mathrm{suf}(s_i), a_{i+1}) = \mathrm{suf}(s_{i+1})$ d'après le Lemme 5.2 et $(\mathrm{suf}(s_1), \ldots, \mathrm{suf}(s_k))$ est aussi un chemin de $\mathfrak{F}(L)$.

Comme le langage $F(L)$ reconnu par $\mathfrak{F}(L)$ est fini, $\mathfrak{F}(L)$ n'a pas de circuits. Les états $\mathrm{suf}(s_1), \ldots, \mathrm{suf}(s_k)$ sont donc deux à deux distincts et $\mathrm{suf}^{-1}(d)$ contient au plus un état de $(s_1, \ldots, s_k)$. $\mathrm{suf}^{-1}(d)$ contient donc au plus $|L|$ liens suffixes sécants. □

**Remarque.** L'hypothèse que le lien suffixe est sécant est essentielle dans le Lemme 8.7 car, si $L$ est réduit à l'unique mot $x = ab_1ab_2 \ldots ab_sa$ avec des lettres $a, b_1, \ldots, b_s$ deux à deux distinctes alors

$$\mathrm{suf}^{-1}([a]) = \{[ab_1a], [ab_1ab_2a], \ldots, [ab_1 \ldots ab_sa]\}.$$

**8.8. Lemme.** *Lors de la construction de $\mathfrak{F}(L)$, un lien suffixe peut être modifié au plus* $2(\mathrm{lgmax} - 1)$ *fois.*

**Preuve.** (i) Si $d_2$ est l'état dupliqué d'un état $d_1$ selon $(r, b)$ et si $u_1$ et $u_2$ sont les mots qui représentent respectivement $d_1$ et $d_2$, alors

$$|u_2| = \mathrm{rg}(d_2) = \mathrm{rg}(r) + 1 < \mathrm{rg}(d_1) = |u_1|.$$

Si cette duplication est frontale et se réalise lors de l'introduction du mot $x_{i_2}$ et si

$i_1 = \text{part}(d_1)$, alors

$$|\text{pref}_{i_2}(u_2)| = |u_2| < |u_1| \leqslant \text{pref}_{i_1}(u_1).$$

(ii) Si un mot $v$ est transféré d'un état $d_1$ à un état $d_2$ alors, avant le transfert, $d_1 = \text{suf}(d_2)$ et $\exists w \in A^*$ tel que les états $f_1 = \tau(d_1, w)$ et $f_2 = \tau(d_2, w)$ soient ultimes avec $i_1 = \text{part}(f_1) < i_2 = \text{part}(f_2)$ et $\text{suf}(f_2) = f_1$ et le transfert est provoqué par la lettre $a$ de $A$ qui crée la transition de $f_2$ vers le nouvel état ultime $\tau(f_2, a)$. Les mots $x_{i_1}$ et $x_{i_2}$ de $L$ sont alors tels que $x_{i_1} = \text{pref}_{i_1}(v)w$ et $x_{i_2} = \text{pref}_{i_2}(v)wa$ et $|x_{i_2}| \leqslant |x_{i_1}|$ implique $|\text{pref}_{i_2}(v)| < |\text{pref}_{i_1}(v)|$.

(iii) Supposons que, lors de la prise en compte d'une lettre $a$ d'un mot $x_{i_2}$ de $L$, un état $d_1$ soit dupliqué en un état $d_2$ avec $i_2 > i_1 = \text{part}(d_1)$ et que les mots $u_1$ et $u_2$ qui représentent respectivement $d_1$ et $d_2$ vérifient $|\text{pref}_{i_2}(u_2)| > |\text{pref}_{i_1}(u_1)|$. D'après (i), une telle duplication n'est pas frontale. Il existe alors un facteur gauche $w_1$ de $w = (\text{pref}_{i_1}(u_1))^{-1}x_{i_1}$ et $w_2 \in A^*$ tels que $x_{i_2} = \text{pref}_{i_2}(u_2)w_1aw_2$ et $|x_{i_2}| \leqslant |x_{i_1}|$ implique $|w_1| + |w_2| < |w|$. En outre, la première lettre $b$ de $w_1^{-1}w$ est distincte de $a$. Si $s_1 = \tau(d_1, w_1)$, alors $\tau(s_1, a) \neq \emptyset$ et $\tau(s_1, b) \neq \emptyset$ et aucun mot de $L$ ne peut provoquer à lui seul un transfert de mots de $d_2$ vers un autre état. Pour qu'un tel transfert puisse exister, il faut que $L$ contienne le mot $x_{i_3} = \text{pref}_{i_2}(u_2)w$ avec $i_3 > i_1$ et $\text{suf}([x_{i_3}]) = [x_{i_1}]$. $|x_{i_3}| \leqslant |x_{i_1}|$ implique alors $|\text{pref}_{i_2}(u_2)| = |\text{pref}_{i_1}(u_1)|$. Un mot $x_{i_4} = u_4'u_4wc$ où $c \in A$, $u_4$ est le plus grand mot de $\text{Fd}(u_1) \cap \text{Fd}(u_4'u_4)$ tel que $|u_4| > |u_2|$ et $i_4 > i_3$ provoque alors le transfert complet de $d_2$ vers un état $d_4$ représenté par $u_4$ et $|x_{i_4}| \leqslant |x_{i_1}|$ implique $|\text{pref}_{i_4}(u_4)| < |\text{pref}_{i_1}(u_1)|$.

(iv) $u$ étant un facteur donné de $L$, soit $d_1, d_2, \ldots, d_k$ la suite des états $\text{suf}([u])$ lors de la construction de $\mathfrak{F}(L)$ et soit $i_1 = \text{part}(d_1)$ et $u_1$ le représentant de $d_1$. Si, $\forall i \in \{1, \ldots, k-1\}$, $d_{i+1}$ est un dupliqué de $d_i$, alors $k < \text{rg}(d_1) \leqslant \text{lgmax}$ d'après (i). Si, $\forall i \in \{1, \ldots, k-1\}$, il y a transfert de mots de $d_i$ à $d_{i+1}$, alors $k < |\text{pref}_{i_1}(u_1)| \leqslant \text{lgmax}$.

D'après (iii), il peut exister au plus $|\text{pref}_{i_1}(u_1)| - 1$ transferts complets concernant $d_1, \ldots, d_k$ et, comme chaque duplication partage le nombre de mots reconnus en deux parties non vides et strictement plus petites, $k \leqslant 2(\text{lgmax} - 1)$. □

**8.9. Théorème.** *La détermination du transducteur de progression* $\mathfrak{F}(L)$ *est en* $O(\|L\| \cdot (|A| + \min(|L|, \text{lgmax})))$ *et en moyenne en* $\mathcal{O}(\|L\| \cdot |A|)$.

**Preuve.** (i) Si $\rho$ et $\delta$ sont respectivement le nombre de transitions créées ou redéfinies et le nombre de duplications et de transferts réalisés lors de la détermination de $\mathfrak{F}(L)$, alors $\delta \leqslant 4\|L\| - 7|L|$ d'après le Lemme 8.2 et, en cumulant la relation du Lemme 8.1 pour toutes les lettres de tous les mots de $L$, $\rho \leqslant (1 + |A|) + 2\|L\|$ d'où $\rho \leqslant (1 + |A|)(4\|L\| - 7|L|) + 2\|L\|$.

(ii) Lors de l'exécution d'une des procédures PARTAGE ou DECPARTAGE au plus $|L|$ liens suffixes sécants sont redéfinis d'après le Lemme 8.7 et au plus $|A|$ liens suffixes non sécants et points d'éclatement le sont d'après le Lemme 8.3. Il résulte alors de (i) que le nombre total de liens suffixes redéfinis est majoré par $4\|L\| \cdot (|L| + |A|)$.

(iii) D'après le Lemme 8.6, $\mathfrak{F}(L)$ admet au plus $\|L\|-2|L|$ liens suffixes sécants et chacun d'eux peut être redéfini au plus $2(\text{lgmax}-1)$ fois d'après le Lemme 8.8. Le nombre total de liens suffixes sécants redéfinis est donc aussi majoré par $(\|L\|-2|L|)\cdot 2(\text{lgmax}-1)$, alors que celui des liens suffixes non sécants le reste par $4\|L\|\cdot|A|$ d'après (ii).

(iv) La complexité de la détermination de $\mathfrak{F}(L)$ dépend aussi du nombre $\|L\|$ de lettres des mots de $L$ et du nombre d'états ultimes créés qui est majoré par $2\|L\|-3|L|+1$ d'après le Théorème 2.3. Il résulte alors de (i), (ii) et (iii) que cette complexité est en $O(\|L\|\cdot(|A|+\min(|L|, \text{lgmax})))$.

(v) Si on suppose que toutes les lettres de $A$ sont équiprobables, la modification des liens suffixes induite par une lettre de $A$ est indépendante en moyenne de cette lettre et, comme suf reste continuellement une application, la taille moyenne des ensembles $\text{suf}^{-1}(d)$ est de 1 pour un langage $L$ donné. En supposant les langages équiprobables le nombre moyen de liens suffixes redéfinis est donc en $\mathcal{O}(\|L\|)$. En tenant compte de la modification des points d'éclatement et des initialisations pour $\tau$ et $\varphi$, la détermination de $\mathfrak{F}(L)$ est donc en $\mathcal{O}(\|L\|\cdot|A|)$. □

**8.10. Remarque.** L'exemple qui suit montre que le nombre de liens suffixes redéfinis n'est pas en $O(\|L\|\cdot|A|)$.

**Exemple.** (i) Soit $A=\{a, b, c_1, \ldots, c_k\}$, $x_1=a^s(a^{r-1}b)^{m+1}$ avec $s>0$, $r\geqslant 1$ et $m\geqslant 0$ et, $\forall i\in\{1,\ldots,q\}$ où $1<q<r+s-1$ et $\forall j\in\{1,\ldots,k\}$ soit $x_{(i-1)k+j+1}=a^{r+s-1-i}c_j a^{i-1}b(a^{r-1}b)^m$. Si $L_1=\{x_1,\ldots,x_{qk+1}\}$, $d_1=[a^{r+s-1}b]=\{a^{r+s-1}b, a^{r+s-2}b, \ldots, b\}$ et, $\forall i\in\{1,\ldots,rm\}$ si $w_i$ est le facteur gauche de $(a^{r-1}b)^m$ de longueur $i$, $d_{1+i}=[a^{r+s-1}bw_i]=\tau([a^{r+s-1}b], w_i)$ sont des états de $\mathfrak{F}(L_1)$ et $\mathfrak{F}(L_1)$ admet $(qk+1)(rm+1)-1$ liens suffixes sécants.

(ii) Si, $\forall i\in\{1,\ldots,p\}$, où $1\leqslant p\leqslant r+s-1-q$, $y_i=a^{r+s-1-i}b(a^{r-1}b)^m a^i$ et si $L_2=\{y_1,\ldots,y_p\}$ et $L=L_1\cup L_2$, l'adjonction de chacun des mots $y_i$ crée une duplication frontale et tous les liens suffixes sécants sont redéfinis lors de cette duplication.

Le nombre total de liens suffixes sécants redéfinis est alors $p[(q^{k+1})(rm+1)-1]$ alors que $|A|\cdot\|L\|=(k+2)(rm+r+s)(qk+1+p)$.

## 9. Le transducteur de réinitialisation de $L$

**9.1. Définition.** (i) $\forall(s, v)\in S\times A^*$ tel que $\tau$ ne soit pas défini en $(s, v)$, tous les mots reconnus par $s$ admettent le même plus grand facteur droit $w$ tel que $wv\in F(L)$. Il existe donc une application $\tau'$ de $S\times A^*$ dans $S$ définie en $(s, a)\in S\times A$ si, et seulement si, $\tau$ n'est pas définie en $(s, a)$ et telle que, $\forall u\in F(L)$ tel que $ua\notin F(L)$, $\tau'([u], a)=[wa]$ où $wa$ est le plus grand mot de $\text{Fd}(ua)\cap F(L)$. $(S, \tau')$ est alors un automate partiel et $\tau'$ est donc défini par sa restriction à $S\times A$.

(ii) Il existe aussi une application $\varphi'$ de $S\times A$ dans $\mathbb{N}$ définie en $(s, a)\in S\times A$ si, et seulement si, $\tau'$ l'est et telle que, si $u$ représente l'état $s$ et si $va$ est le plus grand

mot de $\mathrm{Fd}(ua) \cap F(L)$, $\varphi'(s, a) = \mathrm{pref}(\tau'(s, a)) - |va|$, c'est-à-dire telle que $\varphi'(s, a)$ soit égal au nombre de lettres situées avant la première occurrence de $va$ comme facteur du mot $x_i$ de $L$ tel que $i = \mathrm{part}([va])$.

(iii) $\mathfrak{F}'(L) = (S, \tau', \varphi')$ est alors un transducteur appelé le *transducteur de réinitalisation de L*; $\tau'$ et $\varphi'$ sont respectivement la *fonction de transition* et la *fonction de sortie* de de $\mathfrak{F}'(L)$.

**Exemple.** Lorsque $L = \{x_1, x_2, x_3\}$ avec $x_1 = abbcabc$, $x_2 = abbabc$ et $x_3 = ababcb$ comme $L'$ sur la Fig. 5, le transducteur de réinitialisation $\mathfrak{F}'(L)$ est représenté par la Fig. 11.

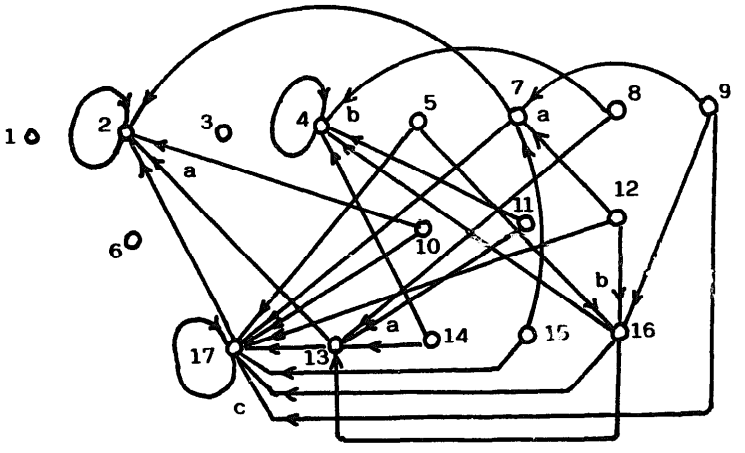


Fig. 11.

**9.2. Lemme.** *Si $\tau$ n'est pas défini en $(s, a) \in S \times A$ et si $(s_0, \ldots, s_k)$ est la suite suffixe de $s$ relative à la lettre $a$ dans $\mathfrak{F}(L)$, alors*

$$\tau'(s, a) = \tau(\mathrm{suf}[s_k], a) \quad et \quad \varphi'(s, a) = \mathrm{avs}(s_k) + \varphi(\mathrm{suf}(s_k), a).$$

**Preuve.** Si, $\forall i \in \{0, \ldots, k\}$, $u_i$ est le mot qui représente $s_i$, alors $u_i a \notin F(L)$ puisque $\tau(s_i, a) = \emptyset$ et $\mathrm{suf}(u_k)$ est donc le plus grand facteur droit $v$ de $u_0$ tel que $va \in F(L)$. Il en résulte que, $\forall i \in \{1, \ldots, k\}$, $\tau'(s_i, a) = \tau(s, a)$ et $\varphi'(s_i, a) = \varphi'(s, a)$, que $\tau'(s, a) = \tau(\mathrm{suf}(s_k), a)$ et que $\varphi'(s, a) = \mathrm{avs}(s_k) + \varphi(\mathrm{suf}(s_k), a)$ d'après la définition de avs et d'après le Lemme 4.2. □

**9.3. Les procédures REINITALISE et TRANSREINITIALISE.** (i) Comme $\tau'$ est défini en $(s, a) \in S \times A$ si, et seulement si, $\tau$ ne l'est pas, il est possible d'utiliser un même tableau pour $\tau$ et $\tau'$ en distinguant $\tau$ et $\tau'$ par leur signe: si $\tau'$ est défini en $(s, a)$, alors $\tau(s, a) = -\tau'(s, a)$.

(ii) La procédure est récursive et est appelée pour $s_0 \in S$ et $c \in A$ tels que $\tau(s_0, c) = \emptyset$ ($\tau(s_0, c) = 0$ dans la procédure). Si $(s_0, \ldots, s_k)$ est la suite suffixe de $s_0$ relative à la lettre $c$, la récursivité se termine par l'appel de la procédure pour $(s_k, c)$ lorsque $\tau(s_k, c) = \emptyset$ et sinon, pour un couple $(s_i, c)$ avec $i \in \{1, \ldots, k-1\}$ pour lequel $\tau'(s_i, c)$ a déjà été calculé.

```
Procedure REINITALISE(s0: integer; c: char);
begin s1 := suf(s0); t1 := τ(s1, c);
  if t1 = 0
  then REINITIALISE(s1, c)
  else
  begin if t1 > 0
    then begin u := −t1; v := avs(s0) + φ(s1, c) end
    else begin u := t1; v := φ(s1, c) end;
  end;
  τ(s0, c) := u; φ(s0, c) := v;
end.
```

(iii) La procédure TRANSREINITIALISE est la procédure d'appel de la procédure REINITIALISE.

```
Procedure TRANSREINITIALISE;
begin for r := 1 to q do
  for k := 1 to carda do
    begin c := let(k); if τ(r, c) = 0 then REINITIALISE(r, c) end;
end.
```

**9.4. Théorème.** *Le transducteur de réinitialisation $\mathfrak{F}'(L)$ est déterminé par la procédure* TRANSREINITIALISE *et son calcul est en* $O(\|L\| \cdot |A|)$.

**Preuve.** (i) D'après le Lemme 9.2, le calcul de $\tau'$ et de $\varphi'$ est correct pour le premier couple $(s_1, a_1) \in S \times A$ pour lequel ils sont calculés et, en supposant les calculs corrects pour les $k$ premiers couples $(s, a)$, il l'est aussi pour le $(k+1)$ième. Il en résulte, par récurrence, que $\mathfrak{F}'(L)$ est correctement calculé.

(ii) Pour chaque couple $(s, a)$ de $S \times A$ tel que $\tau(s, a) = \emptyset$, la procédure REINITIALISE est appelée une unique fois et, comme cette procédure est en $O(1)$, la détermination de $\mathfrak{F}'(L)$ est un $O(|S \times A|)$. D'après le Théorème 2.3, elle est donc aussi $O(\|L\| \cdot |A|)$. □

## 10. La reconnaissance des facteurs de $L$ dans un texte

**10.1. La procédure FACTEURCOMMUN.** Un mot texte suivi du symbole de terminaison \$ est lu lettre par lettre et la procédure FACTEURCOMMUN donnée ci-dessous détermine successivement toutes les occurrences de facteurs de $L$ qui sont maximaux dans le mot texte à l'aide des transducteurs $\mathfrak{F}(L)$ et $\mathfrak{F}'(L)$. Pour chaque occurrence d'un tel facteur $u$, la procédure EDITE est appelée pour imprimer $u$ et sa longueur $|u| = \text{pref}(s) - i - 1$.

```
Procedure FACTEURCOMMUN;
begin s := 1; i := 1; read(c);
```

```
while c<>"$" do
begin
  if c in alpha
  then
  begin t := τ(s, c);
    if t>0
    then begin i := i+φ(s, c); s := t end
    else begin EDITE; i := φ(s, c)+1; s := −t end;
  end
  else begin if s>1 then EDITE; i := 1; s := 1 end;
  read(c);
end;
if s>1 then EDITE;
end.
```

**10.2. Théorème.** *L'algorithme ci-dessus détermine toutes les occurrences de facteurs de L dans le texte en temps linéaire relativement à la longueur du texte et indépendamment de la taille de l'alphabet du texte.*

**Preuve.** (i) Soient $u$ le facteur de $L$ trouvé, $s$ l'état de $\mathfrak{F}(L)$ qui le reconnaît et $c$ la lettre qui suit l'occurrence de $u$ dans le mot texte. On suppose soit que $u$ est un facteur gauche de texte soit que la lettre $a$ de $A$ qui précède cette occurrence de $u$ dans texte est telle que $au \notin F(L)$.

Si $uc \in F(L)$, alors $\tau(s, c) > 0$ et, en substituant $\tau(s, c)$ à $s$, la longueur du facteur commun progresse de 1. Si $i$ était l'indice de la première lettre de $u$, $i+\varphi(s, c)$ est alors l'indice de la première lettre de $uc$.

Si $uc \notin F(L)$, $u$ est une occurrence de facteur de $L$ maximale dans le mot texte. Alors $t = \tau'(s, c) = -\tau(s, c)$ reconnaît le plus grand mot $v$ de $F(L) \cap \mathrm{Fd}(uc)$ d'après 9.1 et $i = \varphi'(s, c)+1$ est l'indice de la première lettre de $v$ dans le mot $x_j$ avec $j = \mathrm{part}(t)$ d'après la Définition 9.1.

(ii) Comme le traitement d'une lettre du texte est en O(1), l'algorithme est bien linéaire en fonction de la longueur du texte. Seules les lettres de l'alphabet du langage $L$ provoquent une transition dans $\mathfrak{F}(L)$ ou dans $\mathfrak{F}'(L)$ mais la réinitialisation dans le cas où une lettre $c$ du texte n'appartient pas à cet alphabet est triviale: l'algorithme est indépendant de l'alphabet du texte. □

**Remarque. 10.3.** Si $m$ est la somme des longueurs des facteurs trouvés, l'édition de ces facteurs est évidemment en $O(m)$.

En outre, si $p$ est le nombre de facteurs trouvés et si $q = \mathrm{lgtexte}-p$ où lgtexte est la longueur du texte, pour tout facteur maximal $u$ trouvé $|u| < \min(q, \mathrm{lgmax})$ car, pour chaque lettre $c$ du texte, ou bien $uc \in F(L)$ et $u$ n'est pas maximal ou bien $u$ est remplacé par $v \in \mathrm{Fd}(uc)$ tel que $|v| \leqslant |u|$. Il en résulte que $m \leqslant \mathrm{lgtexte} \cdot \min(\mathrm{lgmax}, \frac{1}{2}\mathrm{lgtexte})$.

**Exemple.** Si $L=\{x_1, x_2, x_3\}$ avec $x_1 = abbcabc$, $x_2 = abbabc$ et $x_3 = ababcb$, le transducteur de progression $\mathfrak{F}(L)$ est donné par la Fig. 5 après le Théorème 3.4 et le transducteur de réinitialisation $\mathfrak{F}'(L)$ est donné par la Fig. 11 après la Définition 9.1.

La lecture du texte = *babcbabbcabbab* suivi de \$ provoque alors successivement:

- la progression $\tau(1, b)=6$, $\tau(6, a)=13$, $\tau(13, b)=14$, $\tau(14, c)=15$, $\tau(15, b)=16$ stoppée par $\tau(16, a)=-13$ et $i=1+\varphi(1, b)+\varphi(6, a)+\varphi(13, b)+\varphi(14, c)+\varphi(15, b)=2$ déterminent le facteur $u_1 = babcb$ de $x_3$;
- la réinitialisation avec $13=\tau'(16, a)$ et $i=\varphi'(16, a)+1=2$;
- la progression $\tau(13, b)=14$ stoppée par $\tau(14, b)=-4$ et $i=2+\varphi(13, b)=2$ déterminent le facteur $u_2 = bab$ de $x_3$;
- la réinitialisation avec $4=\tau'(14, b)$ et $i=\varphi'(14, b)+1=1$;
- la progression $\tau(4, c)=5$, $\tau(5, a)=7$, $\tau(7, b)=8$ stoppée par $\tau(8, b)=-4$ et $i=1+\varphi(4, c)+\varphi(5, a)+\varphi(7, b)=1$ déterminent le facteur $u_3 = abbcab$ de $x_1$;
- la réinitialisation avec $4=\tau'(8, b)$ et $i=\varphi'(8, b)+1=1$;
- la progression $\tau(4, a)=10$, $\tau(10, b)=11$ stoppée par \$ et $i=1+\varphi(4, a)+\varphi(10, b)=1$ déterminent le facteur $u_4 = abbab$ de $x_2$. Voir Fig. 12.

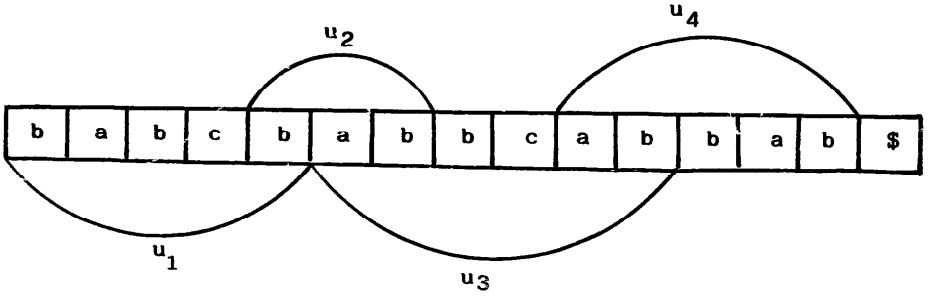


Fig. 12.

**Remarque.** Si un linguiste veut utiliser ce programme pour déterminer toutes les occurrences de mots d'une même famille ou relatifs à un même concept en éliminant en outre les mots paronymes, il suffit d'introduire un tableau booléen qui élimine à l'édition tous les états relatifs à ces mots.

**Conclusion.** Dans la Proposition 1.3, nous avons comparé plusieurs automates qui reconnaissent les facteurs d'un langage fini $L$ et nous avons choisi le plus petit d'entre eux pour développer les algorithmes. Les algorithmes relatifs aux autres sont plus simples car il n'y a plus de transferts et, pour $\tilde{D}=(\mathrm{Fd}(x_1), \ldots, \mathrm{Fd}(x_t))$, il n'y a plus de liens suffixes sécants. La construction en ligne de ces automates est en $O(\|L\| \cdot |A|)$ ce qui montre leur intérêt. En outre, pour chacun la construction de l'automate de réinitialisation reste la même et l'algorithme de reconnaissance des facteurs de $L$ dans un texte reste aussi le même.

## Références


[1] A.V. Aho and M.J. Corasick, Efficient string matching; an aid to bibliographic research, *Comm. ACM* **18**(6) (1975) 333-340.

[2] A.V. Aho, Pattern matching in strings, in: R.V. Book, ed., *Formal Language Theory* (Academic Press, New York, 1980) 325-347.

[3] J. Berstel, *Transductions and Context-free Languages* (Teubner, Stuttgart, 1979).

[4] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler and R. McConnell, Linear size finite automata for the set of all subwords of a word; an outline of results, *Bull. EATCS 21* (1983) 12-20.

[5] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler and R. McConnell, Building the minimal DFA for the set of all subwords of a word on-line in linear time, in: *Proc. ICALP 1984*, Lectures Notes in Computer Sciences **172** (Springer, Berlin, 1984) 109-118.

[6] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler and R. McConnell, Building a complete inverted file for a set of text files in linear time, in: *Proc of 16th ACM Symp. on the Theory of Computing*, ACM, New York (1984) 349-358.

[7] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M.T. Chen and J. Seiferas, The smallest automaton recognizing the subwords of a text, *Theoret. Comput. Sci.* **40** (1985) 31-55.

[8] A. Blumer, J. Blumer, D. Haussler, R. McConnell and A. Ehrenfeucht, Complete inverted files for efficient text retrieval and analysis, *J. ACM* **34**(3) (1987) 578-595.

[9] R.S. Boyer and J.S. Moore, A fast string searching algorithm, *Comm. ACM* **20**(10) (1977) 762-772.

[10] E.M. McCreight, A space-economical suffix-tree construction algorithm, *J. ACM* **23**(2) (1976) 262-272.

[11] M.T. Chen and J. Seiferas, Efficient and elegant subword-tree construction, in: *Proc. NATO Advanced Research Workshop on Combinatorial Algorithms on Words*, Maratea, Italy (1984) 97-107.

[12] M. Crochemore, Optimal factor transducers, in: *Proc. NATO Advanced Research Workshop on Combinatorial Algorithms on Words*, Maratea, Italy (1984) 31-43.

[13] M. Crochemore, Transducers and repetitions, *Theoret. Comput. Sci.* **45**(1) (1986) 63-86.

[14] M. Crochemore, Computing LCF in linear time, *Bull. EATCS 30* (1986) 57-61.

[15] S. Eilenberg, *Automata, Languages and Machines* (Academic Press, New York, 1974).

[16] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern-matching in strings, *SIAM J. Comput.* **6**(2) (1977) 323-350.

[17] M. Lothaire, *Combinatorics on Words* (Addison-Wesley, Reading, MA, 1983).

[18] J.H. Morris and V.R. Pratt, A linear pattern-matching algorithm, Tech. Rept. 40, Computing Center, University of California, Berkeley, CA (1970).

[19] A. Nerode, Linear automaton transformations, *Proc. Amer. Math. Soc.* **9** (1958) 541-544.

[20] J.-C. Spehner, La reconnaissance des facteurs d'un mot dans un texte, *Theoret. Comput. Sci.* **48** (1986) 35-52.

[21] J.-C. Spehner, Sur les automates qui reconnaissent une famille de langages, Publication du Lab. Math-Info No. 46, Université de Haute Alsace, Mulhouse, France.

[22] P. Weiner, Linear pattern-matching algorithms, in: *Proc. 14th IEEE Ann. Symp. on Switching and Automata Theory*, Iowa, U.S.A. (1983) 1-11.